
Aplicación Android para gestionar la dieta de un deportista

Por
David Fernández Alejo
Vitaliy Savchenko
Carlos Segundo Nieto
Víctor Velasco Arjona



UNIVERSIDAD COMPLUTENSE MADRID

Grado en Ingeniería de Software
FACULTAD DE INFORMÁTICA

Dirigido por
Antonio Sarasa Cabezuelo

Android application to manage the diet of an
athlete

MADRID, 2021–2022

Aplicación Android para gestionar la dieta de un deportista

Android application to manage the diet of an athlete

Memoria que se presenta para el Trabajo de Fin de Grado

David Fernández Alejo
Vitaliy Savchenko
Carlos Segundo Nieto
Víctor Velasco Arjona

Dirigido por

Antonio Sarasa Cabezuelo

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Madrid, 2022

Agradecimientos

Agradecemos a nuestras familias el apoyo incondicional durante la realización del trabajo y a todas las personas que nos han dado una opinión crítica de la aplicación, ayudando a mejorarla.

Resumen

En este proyecto se describe la especificación de requisitos, diseños e implementación de una aplicación Android para la gestión de dietas para deportistas.

Todo deportista necesita complementar su actividad física con una dieta saludable. En este sentido, un problema bastante habitual es ser constante en el seguimiento de la misma. Es por ello que este trabajo se ha tratado de resolver este problema proporcionando una aplicación orientada específicamente a gestionar dietas y a facilitar al usuario el seguimiento de las mismas. La aplicación ofrece al usuario la posibilidad de crear y gestionar una serie de dietas para su seguimiento. También se brinda la posibilidad de publicar las dietas creadas por un usuario, las cuales serán visibles por otros usuarios, creando así una red de personas que pueden comentar y valorar las dietas que siguen, para dejar constancia de su eficacia y ayudar con ello al resto de personas que usan la aplicación.

Durante el seguimiento de una dieta, el usuario puede introducir la cantidad consumida de cada uno de los alimentos que componen la dieta, generando un registro de la ingesta calórica por cada día de la semana. Como complemento a todo ello, también se puede introducir la cantidad de pasos caminados durante cada día, así como el peso para que el usuario pueda ver una representación gráfica del progreso.

Palabras clave

Android, Spring, sports, API, Node.js

Abstract

This project describes the requirements specification, designs and implementation of an Android application for diet management for athletes.

Every athlete needs to complement their physical activity with a healthy diet. In this sense, a fairly common problem is to be consistent in monitoring it. That is why this work has tried to solve this problem by providing an application specifically oriented to manage diets and to make it easier for the user to follow them. The application offers the user the possibility of creating and managing a series of diets for follow-up. It also offers the possibility of publishing the diets created by a user, which will be visible to other users, thus creating a network of people who can comment on and evaluate the diets they follow, to record their effectiveness and thus help the rest of the people who use the application.

While following a diet, the user can enter the amount consumed of each of the foods that make up the diet, generating a record of caloric intake for each day of the week. As a complement to all this, the amount of steps walked during each day can also be entered, as well as the weight so that the user can see a graphical representation of the progress.

Keywords

Android, Spring, Sports, API, Node.js

Índice general

| | Página |
|---|-----------|
| A. Contenido de la memoria | 1 |
| 1. Introducción | 1 |
| 1.1. Motivación | 1 |
| 1.2. Objetivos | 1 |
| 1.3. Organización de la memoria | 2 |
| 1.4. Planificación | 3 |
| 1. Introduction | 1 |
| 1.1. Motivation | 1 |
| 1.2. Objectives | 1 |
| 1.3. Memory organization | 2 |
| 1.4. Planning | 2 |
| 2. Estado de la cuestión | 4 |
| 3. Tecnología empleada | 8 |
| 3.1. Herramientas utilizadas en la parte del cliente | 8 |
| 3.1.1. Android Studio | 8 |
| 3.2. Herramientas utilizadas en la parte del servidor | 8 |
| 3.2.1. Google Firebase | 8 |
| 3.2.2. Spring Boot | 10 |
| 3.2.3. Node.js | 10 |
| 3.2.4. Retrofit | 11 |
| 3.3. Otras herramientas | 11 |
| 3.3.1. Git | 11 |
| 3.3.2. Trello | 11 |
| 3.3.3. Overleaf | 12 |
| 3.3.4. Google Drive | 12 |
| 3.3.5. Visual Studio Code | 12 |
| 4. Casos de uso | 13 |
| 4.1. Actores del sistema | 13 |
| 4.2. Módulo deportista | 13 |
| 4.3. Módulo administrador | 29 |

| | |
|---|------------|
| 5. Arquitectura | 37 |
| 5.1. Arquitectura de la aplicación | 37 |
| 5.2. Patrones de diseño y arquitectónicos | 38 |
| 5.2.1. Patrones arquitectónicos | 38 |
| 5.2.2. Patrones de diseño | 39 |
| 6. Modelo de datos | 42 |
| 6.1. Modelo E-R | 42 |
| 6.2. Implementación de la base de datos | 43 |
| 6.2.1. Colección de documentos de usuarios | 43 |
| 6.2.2. Colección de documentos de dietas | 45 |
| 6.2.3. Colección de documentos de alimentos | 46 |
| 6.2.4. Colección de documentos de comentarios | 47 |
| 6.2.5. Colección de documentos de pasos | 47 |
| 6.2.6. Colección de documentos de pesos | 48 |
| 6.2.7. Colección de documentos de historial de dietas | 49 |
| 6.3. Almacenamiento de archivos | 50 |
| 7. Diseño de la aplicación | 52 |
| 7.1. Colores de la aplicación | 52 |
| 7.2. Funcionalidad del cliente | 52 |
| 7.2.1. Ciclo de vida de una actividad | 53 |
| 7.2.2. Iniciar sesión | 57 |
| 7.2.3. Crear dieta y modificarla | 67 |
| 7.2.4. Dieta actual | 86 |
| 7.2.5. Añadir comentarios | 92 |
| 7.2.6. Perfil del usuario | 96 |
| 8. Evaluación | 113 |
| 8.1. Diseño de la evaluación | 113 |
| 8.2. Resultados de la evaluación | 114 |
| 9. Conclusiones y trabajo futuro | 121 |
| 9.1. Conclusiones | 121 |
| 9.2. Trabajo futuro | 121 |
| 9. Conclusions and future work | 124 |
| 9.1. Conclusions | 124 |
| 9.2. Future work | 124 |
| 10. Aportaciones individuales | 126 |
| 10.1. David Fernández Alejo | 126 |
| 10.2. Carlos Segundo Nieto | 127 |
| 10.3. Vitaliy Savchenko | 128 |
| 10.4. Víctor Velasco Arjona | 129 |
| 11. Bibliografía y enlaces de referencia | 131 |

| | |
|---|------------|
| B. Anexos | 136 |
| I. Guía del usuario | 136 |
| 2.1. Usuarios | 138 |
| 2.1.1. Mis dietas creadas | 138 |
| 2.1.2. Dietas publicadas | 141 |
| 2.1.3. Dieta seguida | 142 |
| 2.1.4. Ver perfil | 143 |
| 2.2. Administradores | 144 |
| 2.2.1. Crear usuario | 144 |
| 2.2.2. Todos los usuarios | 145 |
| 2.2.3. Crear dieta | 146 |
| 2.2.4. Todas las dietas del sistema | 147 |
| 2.2.5. Mis dietas creadas | 148 |
| 2.2.6. Ver perfil | 148 |
| II. Preguntas de la evaluación | 150 |

Índice de figuras

| | |
|--|----|
| 1.1. Diagrama de Gantt de la planificación del proyecto | 3 |
| 1.2. Gantt chart of project planning | 3 |
| 2.1. Menú principal de la aplicación Indya | 4 |
| 2.2. Diferentes vistas de la aplicación Oorenji | 5 |
| 2.3. Distintas pantallas de la aplicación FatSecret | 6 |
| 2.4. Distintas pantallas de la aplicación Nootric | 6 |
| 2.5. Distintas pantallas de la aplicación Fitia | 7 |
| 4.1. Diagrama de casos de uso asociado al actor <i>Deportista</i> | 14 |
| 4.2. Diagrama de casos de uso asociado al actor <i>Administrador</i> | 29 |
| 5.1. Esquema de la arquitectura de la aplicación | 37 |
| 5.2. Esquema de Modelo Vista Presentador | 38 |
| 5.3. Esquema del patrón Singleton | 39 |
| 5.4. Esquema del patrón Façade | 39 |
| 5.5. Esquema del patrón Adapter | 40 |
| 5.6. Esquema del patrón Transfer | 40 |
| 5.7. Esquema del patrón View Helper | 41 |
| 6.1. Diagrama Entidad Relación | 43 |
| 6.2. Colección de documentos de los usuarios | 44 |
| 6.3. Entrada de la colección de documentos de dietas | 46 |
| 6.4. Entrada de la colección de documentos de alimentos | 47 |
| 6.5. Entrada de la colección de documentos de comentarios | 47 |
| 6.6. Entrada de la colección de documentos de pasos | 48 |
| 6.7. Entrada de la colección de documentos de pesos | 49 |
| 6.8. Entrada de la colección de documentos historial de dietas | 49 |
| 6.9. Vista detallada de un documento de una dieta | 50 |
| 6.10. Vista detallada de la imagen de perfil de un usuario | 51 |
| 7.1. Flujo de vida de una actividad | 53 |
| 7.2. Código de la función <code>onCreate()</code> | 54 |
| 7.3. Elementos de la vista Dieta Actual | 55 |
| 7.4. Código de la función <code>getDiet()</code> | 56 |
| 7.5. Código de la función <code>getDietInfo()</code> | 57 |
| 7.6. Pantalla de login en la aplicación DietNow | 58 |
| 7.7. Lista de idiomas soportados por la aplicación DietNow | 59 |
| 7.8. Fragmento del código del fichero con texto traducido | 59 |

| | | |
|-------|--|----|
| 7.9. | Formulario de registro en la aplicación DietNow | 60 |
| 7.10. | Fragmento de código de la creación de la actividad Register | 61 |
| 7.11. | Fragmento de código del botón registrar del formulario | 62 |
| 7.12. | Selector de género en el formulario de registro | 63 |
| 7.13. | Código de la función <code>register()</code> de <code>UserService</code> | 63 |
| 7.14. | Código de la función <code>register()</code> del formulario | 64 |
| 7.15. | Código la consulta a la base de datos | 65 |
| 7.16. | Código de la función que actualiza la vista | 65 |
| 7.17. | Código de <code>UserPageActivity()</code> | 66 |
| 7.18. | Vista del menú principal de un usuario | 67 |
| 7.19. | Código del evento asociado al botón “Mis dietas creadas“ | 68 |
| 7.20. | Código de la actividad <code>MyDietsActivity</code> | 68 |
| 7.21. | Vista “Mis dietas” | 69 |
| 7.22. | Código de la función que carga las dietas creadas | 70 |
| 7.23. | Código de la función asociada al botón + de Mis Dietas | 70 |
| 7.24. | Vista de crear nueva dieta | 71 |
| 7.25. | Código de la función asociada al botón “Guardar“ de crear nueva dieta | 72 |
| 7.26. | Vista de Ver Dieta | 73 |
| 7.27. | Vista de Editar Dieta | 74 |
| 7.28. | Popup que aparece para elegir como añadir un alimento | 75 |
| 7.29. | Vista de añadir un alimento manualmente | 76 |
| 7.30. | Código asociado a añadir alimentos mediante código de barras | 77 |
| 7.31. | Código de <code>GetProductInfo()</code> | 77 |
| 7.32. | Código de <code>getInfo()</code> dentro de <code>GetProductInfo()</code> | 78 |
| 7.33. | Código de la interfaz de la API OpenFoodFacts | 78 |
| 7.34. | Creación del servidor de node.js | 79 |
| 7.35. | Código de la estructura de las peticiones a la API | 80 |
| 7.36. | Código del formateo de los JSON devueltos por OpenFoodFacts | 81 |
| 7.37. | Código del método que añade alimento de forma manual | 82 |
| 7.38. | Añadir alimento mediante el uso de la cámara | 83 |
| 7.39. | Código de <code>CameraActivity</code> | 84 |
| 7.40. | Código del método abre los archivos del teléfono para obtener los documentos | 85 |
| 7.41. | Vista del almacenamiento del dispositivo | 85 |
| 7.42. | Código de <code>DietInfoActivity</code> | 86 |
| 7.43. | Vista Información de la dieta | 87 |
| 7.44. | Código del método <code>getAliments()</code> | 88 |
| 7.45. | Código del adapter <code>DietFollowAdapter()</code> | 88 |
| 7.46. | Código de <code>OnBindViewHolder()</code> | 89 |
| 7.47. | Código del <i>listener</i> del <code>checkBox</code> de cada alimento | 90 |
| 7.48. | Vista de información de dieta con los pasos para actualizar gramos consumidos | 91 |
| 7.49. | Código del método <code>guardar()</code> que almacena los cambios en la base de datos | 91 |
| 7.50. | Vista del módulo comentarios | 92 |
| 7.51. | Código del método <code>comment()</code> que redirige a la vista de los comentarios | 93 |
| 7.52. | Código de la clase <code>DietComments()</code> | 93 |
| 7.53. | Código del <code>getComments()</code> | 94 |
| 7.54. | Código del método <code>OnBindViewHolder()</code> del adaptador <code>CommentsAdapter</code> | 95 |
| 7.55. | Código del método <code>uploadComment()</code> | 95 |

| | |
|---|-----|
| 7.56. Código de los métodos asociados a editar y borrar comentario | 96 |
| 7.57. Vista de Mi Perfil dividida en 3 secciones | 98 |
| 7.58. Código de “UserProfileActivity” | 99 |
| 7.59. Selector de pasos o pesos | 100 |
| 7.60. Código del método AddStep() | 101 |
| 7.61. Consulta que obtiene la información personal de un usuario | 102 |
| 7.62. Código que abre la galería del dispositivo | 102 |
| 7.63. Vista de Gráfico de la dieta actual | 103 |
| 7.64. Layout del botón selector | 104 |
| 7.65. Diseño del thumb del botón selector | 104 |
| 7.66. Diseño del track del botón selector | 105 |
| 7.67. Código del botón selector de gráfica | 105 |
| 7.68. Código del gráfico de pasos | 106 |
| 7.69. Vista “Mi Perfil“ con la lista de acciones desplegada | 107 |
| 7.70. Código de la lista de acciones | 108 |
| 7.71. Vista Histórico de dietas | 109 |
| 7.72. Código del método DietHistory() que genera la vista Histórico de dietas | 110 |
| 7.73. Código del método logout() que genera cierra sesión y redirige al login . | 110 |
| 7.74. Vista de editar perfil | 111 |
| 7.75. Código del método editProfile() | 111 |
| 7.76. Código del método delete() que deshabilita el perfil | 112 |
| 8.1. Resultado de la pregunta de género | 114 |
| 8.2. Resultado de la pregunta de la dificultad de registro | 115 |
| 8.3. Resultado de la pregunta de página principal intuitiva | 115 |
| 8.4. Resultado de la pregunta de información en el perfil | 116 |
| 8.5. Resultado de la pregunta de creación de dietas | 116 |
| 8.6. Resultado de la pregunta del proceso de creación de una dieta | 117 |
| 8.7. Resultado de la pregunta de mejoras en la aplicación | 117 |
| 8.8. Resultado de la pregunta de seguimiento de una dieta | 118 |
| 8.9. Resultado de la pregunta de inserción de alimentos en una dieta | 118 |
| 8.10. Resultado de la pregunta de utilidad de los comentarios | 119 |
| 8.11. Resultado de la pregunta de mejoras en seguimiento de la dieta | 119 |
| 8.12. Resultado de la pregunta de valoración general de la aplicación | 120 |
| 8.13. Resultado de la pregunta de funcionalidades extra | 120 |
| 9.1. Dispositivos utilizados para iniciar sesión en Facebook en Enero de 2022 [1] | 122 |
| 9.1. Devices used to log in to Facebook in January 2022 [1] | 125 |
| 2.1. Vistas de la aplicación | 137 |
| 2.2. Menú principal de la aplicación | 138 |
| 2.3. Vista detallada de un documento de una dieta | 139 |
| 2.4. Una dieta publicada y despública | 140 |
| 2.5. Vista de añadir alimentos a una dieta | 141 |
| 2.6. Vista dietas publicadas | 142 |
| 2.7. Vista de dieta seguida | 143 |
| 2.8. Vista del perfil de usuario | 144 |
| 2.9. Creación de un usuario en la aplicación | 145 |
| 2.10. Vista de todos los usuarios | 146 |

| | |
|---|-----|
| 2.11. Crear una dieta | 147 |
| 2.12. Vista de todas las dietas del sistema | 148 |
| 2.13. Vista de ver perfil de un administrador | 149 |
| 3.1. Preguntas de valoración del usuario | 151 |
| 3.2. Preguntas de valoración de creación de dietas | 152 |
| 3.3. Preguntas de valoración de dieta seguida | 152 |
| 3.4. Preguntas de comentarios sobre la aplicación | 153 |
| 3.5. Preguntas de valoración final de la aplicación | 153 |

Índice de tablas

| | |
|--|----|
| 4.1. Registrar deportista | 14 |
| 4.2. Autenticar deportista | 15 |
| 4.3. Cerrar la sesión de un deportista | 15 |
| 4.4. Modificar datos de un deportista | 16 |
| 4.5. Eliminar perfil de un deportista | 16 |
| 4.6. El deportista crea una dieta | 17 |
| 4.7. Subir documentos a una dieta | 17 |
| 4.8. Eliminar documentos de una dieta | 18 |
| 4.9. Descargar documentos de una dieta | 18 |
| 4.10. Suscribirse a una dieta | 18 |
| 4.11. Modificar una dieta creada por el deportista | 19 |
| 4.12. Ver historial de dietas seguidas | 19 |
| 4.13. Eliminar una dieta creada por del deportista | 20 |
| 4.14. Listar dietas creadas por del deportista | 20 |
| 4.15. Ver todas las dietas publicadas del sistema | 20 |
| 4.16. Buscar dietas | 21 |
| 4.17. Publicar una dieta creada por el deportista | 21 |
| 4.18. Despublicar una dieta creada por el deportista | 22 |
| 4.19. Añadir alimentos a una dieta | 22 |
| 4.20. Eliminar alimentos de una dieta | 23 |
| 4.21. Editar alimentos de una dieta | 23 |
| 4.22. Ver información nutricional de los alimentos a una dieta | 24 |
| 4.23. Informe gráfico de la dieta seguida | 24 |
| 4.24. Añadir comentarios a la dieta seguida | 25 |
| 4.25. Editar comentarios a la dieta seguida | 25 |
| 4.26. Eliminar comentarios a la dieta seguida | 25 |
| 4.27. Añadir pasos caminados diariamente | 26 |
| 4.28. Añadir peso diario | 26 |
| 4.29. El deportista actualiza la información de la dieta seguida | 27 |
| 4.30. El deportista valora la dieta seguida | 27 |
| 4.31. El deportista deja de seguir la dieta | 28 |
| 4.32. Registrar manualmente un usuario | 29 |
| 4.33. Modificar los datos de un usuario | 30 |
| 4.34. Eliminar a un usuario del sistema | 30 |
| 4.35. Autenticar administrador | 31 |
| 4.36. Cerrar la sesión de un administrador | 31 |
| 4.37. El administrador crea una nueva dieta | 32 |
| 4.38. Eliminar una dieta del sistema | 32 |

| | |
|---|----|
| 4.39. Modificar una dieta del sistema | 33 |
| 4.40. Publicar una dieta del sistema | 33 |
| 4.41. Despublicar una dieta del sistema | 34 |
| 4.42. Modificar comentarios de una dieta | 34 |
| 4.43. Eliminar comentarios de una dieta | 35 |
| 4.44. Eliminar perfil de un administrador | 35 |
| 4.45. Listar dietas creadas por del administrador | 35 |
| 4.46. Ver todas las dietas publicadas del sistema | 36 |
| 4.47. Buscar dietas | 36 |

Parte A

Contenido de la memoria

Capítulo 1

Introducción

En este capítulo se explica la motivación, los objetivos, la estructura de trabajo y la planificación del mismo que se ha llevado a cabo para la realización de este proyecto.

1.1. Motivación

Todo deportista necesita acompañar su actividad con una dieta adecuada ya que la alimentación es uno de los factores de los que depende el rendimiento físico. Una dieta adecuada aporta los nutrientes necesarios para mantener un estado óptimo de salud, lo que se traduce en rendimiento. Según cómo se alimente un deportista, podrá ver como afecta la alimentación a su desempeño, mejorando el rendimiento y la recuperación, limitándolos o incluso disminuyéndolos, ya que una mala alimentación puede favorecer lesiones y la fatiga.

Actualmente, no existen aplicaciones móviles que gestionen de forma flexible las dietas que un deportista necesita seguir para llevar una alimentación sana y adecuada a su perfil. Además, las dietas existentes en internet, en su mayoría carecen de fuentes o estudios que las respalden y tampoco tienen un *feedback* fiable por parte de usuarios que la han probado.

Para resolver estas limitaciones, se ha decidido crear una aplicación Android que cubre las necesidades del seguimiento y control de dietas. También se puede ver el *feedback* de los deportistas que han seguido y valorado la dieta, haciendo el papel de red social y ayudando a otros deportistas que busquen objetivos similares. Además, los deportistas pueden adjuntar documentos a las dietas para aportar información adicional que las respalden.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación Android que ayude a los deportistas a complementar su actividad física, siguiendo una dieta sana y equilibrada para conseguir la forma física deseada. Además se puede interactuar con otros deportistas, creando dietas que pueden ser seguidas por otros deportistas o bien incluyendo comentarios y valoraciones en las mismas.

A partir del objetivo principal se pueden definir objetivos más específicos:

- Permitir a un deportista registrarse en la aplicación para descubrir y analizar todas las dietas de la aplicación, pudiendo además actualizar el peso e insertar los pasos diarios, con el objetivo de llevar un recuento y poder visualizar el progreso en gráficas.
- Permitir a los deportistas seguir una dieta insertada desde la aplicación por cualquiera de los usuarios, ya sean deportistas o administradores.
- Ofrecer la posibilidad al deportista de valorar la dieta que está siguiendo, con el fin de ayudar al resto de deportistas a decidir y saber que dietas están siendo efectivas.
- Permitir a todos los deportistas la capacidad de crear una o más dietas, detallando los alimentos que la componen, la cantidad recomendada de cada uno de ellos e incluso publicando documentación que ayude a seguir la dieta.
- Permitir a los deportistas añadir alimentos a las dietas y tener la información nutricional de estos actualizada en todo momento.

1.3. Organización de la memoria

A continuación se describe de manera breve la estructura de la memoria:

- **Capítulo 1:** en este capítulo se describe la motivación del trabajo, los objetivos y la estructura de la memoria.
- **Capítulo 2:** en este capítulo se estudian herramientas similares a la que se ha realizado en el trabajo.
- **Capítulo 3:** en este capítulo se describe la tecnología utilizada para implementar el proyecto.
- **Capítulo 4:** en este capítulo se definen los actores y casos de uso que se explicarán mediante tablas junto a sus requisitos.
- **Capítulo 5:** en este capítulo se explica el diagrama entidad relación en el que se ha basado el proyecto y la implementación de la base de datos.
- **Capítulo 6:** en este capítulo se explica la arquitectura de la aplicación y los patrones utilizados.
- **Capítulo 7:** en este capítulo se realiza un estudio sobre el diseño e implementación de los casos de uso más relevantes.
- **Capítulo 8:** en este capítulo se exponen las estadísticas aportadas por usuarios que han dado su opinión sobre el producto.
- **Capítulo 9:** en este capítulo se enumeran los casos de uso que se harán en un futuro explicándolos brevemente.
- **Capítulo 10:** en este capítulo se expone el trabajo realizado por cada uno de los autores.
- **Anexo I:** manual de usuario.

- **Anexo II:** preguntas de evaluación.

1.4. Planificación

En esta parte se describe toda la planificación llevada a cabo durante el desarrollo de este proyecto, detallando las diferentes reuniones e iteraciones realizadas. En la figura 1.2 se puede apreciar el diagrama de **Gantt** donde se indican las diferentes fases del proyecto.

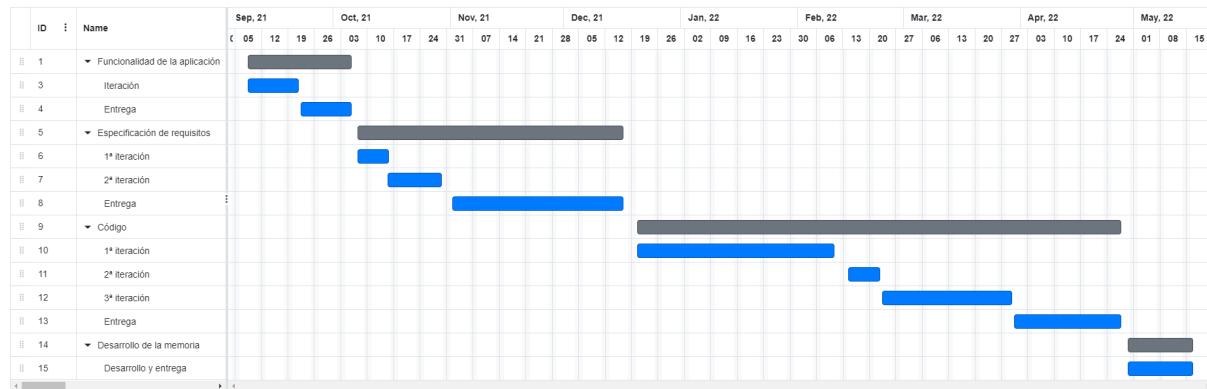


Figura 1.1: Diagrama de Gantt de la planificación del proyecto

Durante la primera fase se establecieron las principales funcionalidades de la aplicación. Una vez terminada dicha fase, se especificaron los requisitos así como los actores y módulos que componen la aplicación. Para ello se realizaron dos iteraciones con el fin de establecer los requisitos finales.

Posteriormente, se establecieron tres iteraciones de código, en cada una de ellas se desarrollaron los distintos módulos previamente establecidos (login, dieta , dieta actual). Tras realizar las iteraciones de código, se realizaron diferentes pruebas de caja blanca sobre los distintos casos de uso. En la última fase se desarrolló la memoria que complementa al código.

Para poder llevar a cabo las diferentes fases mencionadas anteriormente, el equipo realizó reuniones cada sábado, con el fin de establecer la funcionalidad que se iba a desarrollar durante la semana. A su vez las reuniones ayudaron a conocer que habían realizado los integrantes durante la semana anterior, así como intentar resolver de manera conjunta los problemas que iban surgiendo.

Chapter 1

Introduction

This chapter will explain the motivation, objectives, work structure and work planning for this project.

1.1. Motivation

Every athlete needs to accompany his or her activity with an adequate diet, since nutrition is one of the factors on which physical performance depends. An adequate diet provides the necessary nutrients to maintain an optimal state of health, which translates into performance. Depending on how an athlete eats, you can see how food affects their performance, improving performance and recovery, limiting or even decreasing them, as poor nutrition can promote injuries and fatigue.

Currently, there are no mobile applications that can flexibly manage the diets that an athlete needs to follow in order to have a healthy diet that is appropriate to his or her profile. In addition, most of the existing diets on the Internet lack sources or studies that support them and do not have reliable feedback from users who have tried them.

To solve these limitations, it has been decided to create an Android application that meets the needs of monitoring and control of diets. It is also possible to see the feedback of the athletes who have followed and evaluated the diet, playing the role of a social network and helping other athletes who are looking for similar goals. In addition, athletes can attach documents to the diets to provide additional information to support them.

1.2. Objectives

The main objective of this project is to develop an Android application that helps athletes to complement their physical activity, following a healthy and balanced diet to achieve the desired physical shape. It is also possible to interact with other athletes, creating diets that can be followed by other athletes or including comments and evaluations in them.

More specific objectives can be defined based on the main objective:

- Allow an athlete to register in the application to discover and analyze all the diets in the application, and to update the weight and insert the daily steps, in order to

keep a tally and visualize the progress in graphs.

- Allow athletes to follow a diet inserted from the application by any of the users, whether they are athletes or administrators.
- To offer the possibility to the athlete to evaluate the diet he/she is following, in order to help other athletes to decide and know which diets are being effective.
- Allow all athletes the ability to create one or more diets, detailing the foods that compose it, the recommended amount of each one of them and even publishing documentation that helps to follow the diet.
- Allow athletes to add foods to diets and have their nutritional information updated at all times.

1.3. Memory organization

The following is a brief description of the structure of the report:

- **Chapter 1:** this chapter describes the motivation for the work, the objectives and the structure of the report.
- **Chapter 2:** this chapter studies tools similar to the one used in this work.
- **Chapter 3:** this chapter describes the technology used to implement the project.
- **Chapter 4:** this chapter defines the actors and use cases that are explained in tables together with their requirements.
- **Chapter 5:** this chapter explains the entity-relationship diagram on which the project and the implementation of the database were based.
- **Chapter 6:** this chapter explains the architecture of the application and the patterns used.
- **Chapter 7:** this chapter studies the design and implementation of the most relevant use cases.
- **Chapter 8:** this chapter presents the statistics provided by users who have given their opinion on the product.
- **Chapter 9:** this chapter lists cases of uses that will be made in the future, explaining them briefly.
- **Chapter 10:** this chapter describes the work carried out by each of the authors.
- **Annex I:** user's manual.
- **Annex II:** assessment questions.

1.4. Planning

This part describes all the planning carried out during the development of this project, detailing the different meetings and iterations performed. In figure 1.2 the following *Gantt* diagram can be seen where the different phases of the project are indicated.

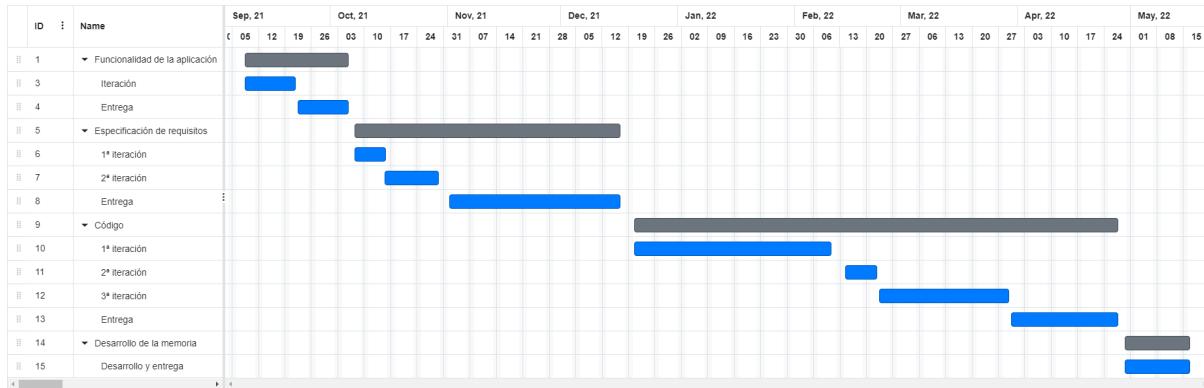


Figura 1.2: Gantt chart of project planning

During the first phase, the main functionalities of the application were established. Once this phase was completed, the requirements were specified as well as the actors and modules that make up the application. Two iterations were carried out in order to establish the final requirements.

Subsequently, three code iterations were established, in each of which the different modules previously established (login, diet, current diet) were developed. After the code iterations, different white box tests were performed on the different use cases. In the last phase, the memory that complements the code was developed.

In order to carry out the different phases mentioned above, the team held meetings every Saturday, in order to establish the functionality to be developed during the week. At the same time, the meetings helped to know what the team members had done during the previous week, as well as to try to solve together the problems that arose.

Capítulo 2

Estado de la cuestión

En este capítulo se describen las características más importantes acerca de otros proyectos que se utilizan actualmente para la gestión de dietas.

Indya

Es una aplicación [2] (ver figura 2.1) para iOS y Android que genera un menú semanal completo para seguir durante un mes. A su vez ayuda a realizar la lista de la compra, indicando los ingredientes y calorías que se necesitan cada semana para rendir al máximo. La aplicación cuenta a su vez con un nutricionista, que recibe los datos de un usuario y reajusta su planificación para conseguir los objetivos.

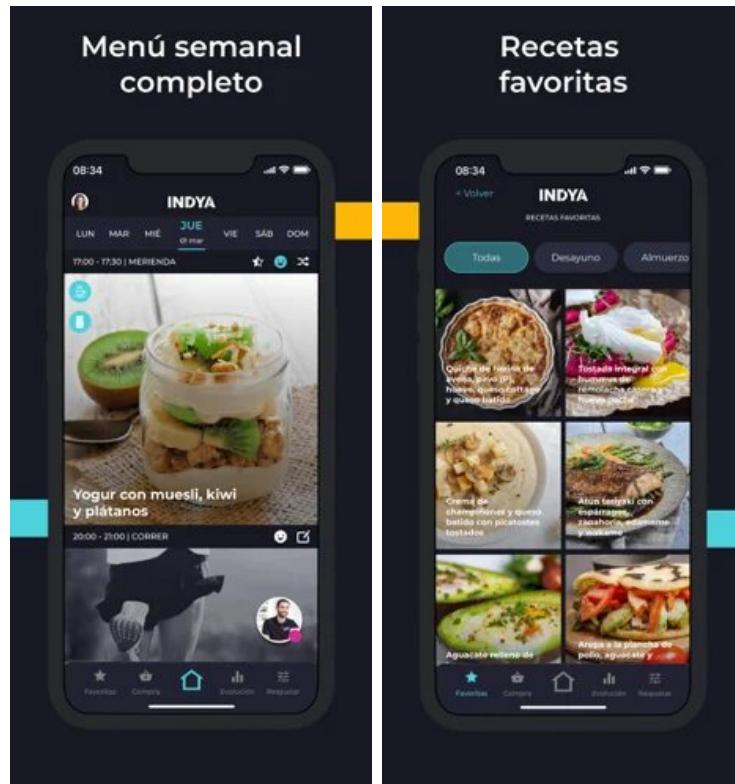


Figura 2.1: Menú principal de la aplicación **Indya**

Oorenji

Es una aplicación [3] (ver figura 2.2) donde a partir de los gustos, alergias, del perfil físico y psicológico del usuario, genera una dieta totalmente adaptada para poder conseguir todos los objetivos. Además se puede contactar con un nutricionista de forma *online* para obtener asistencia. La aplicación propone además una serie de retos para ayudar a cumplir los objetivos.

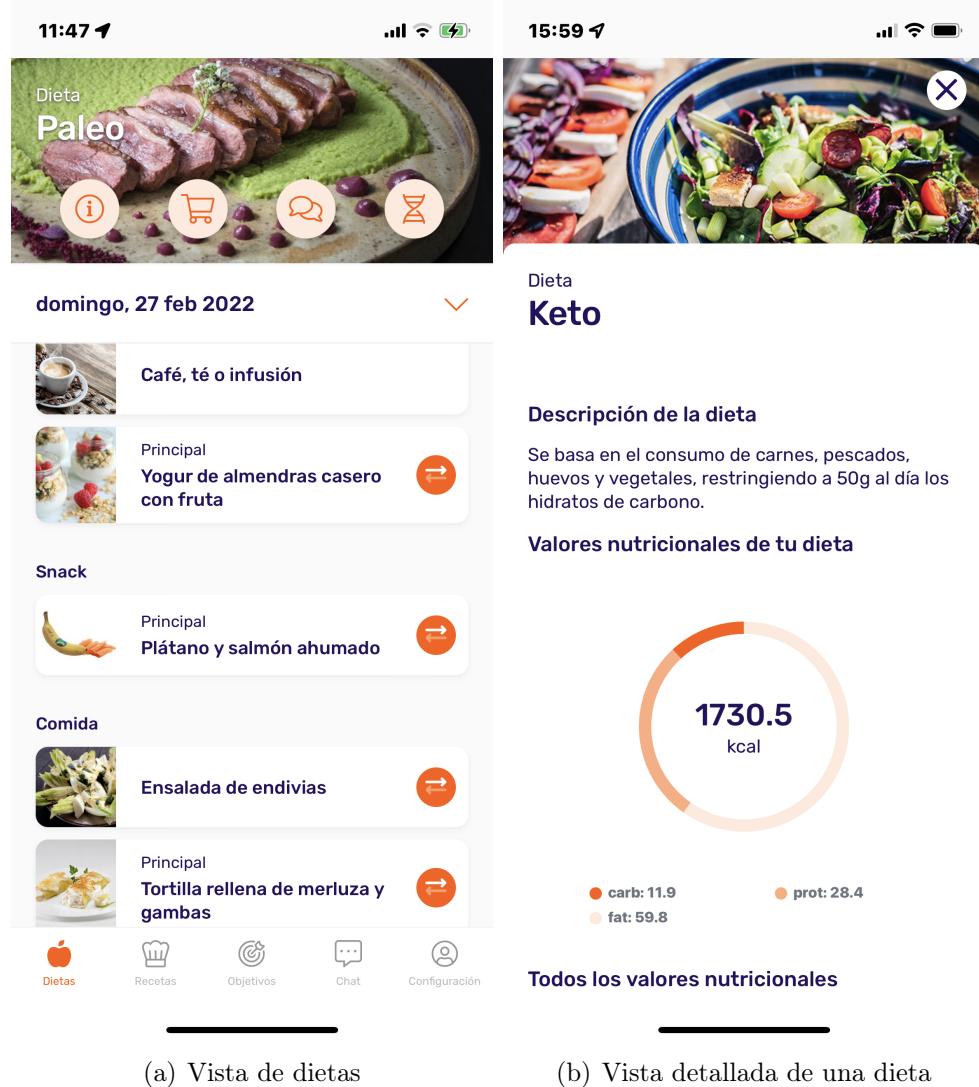
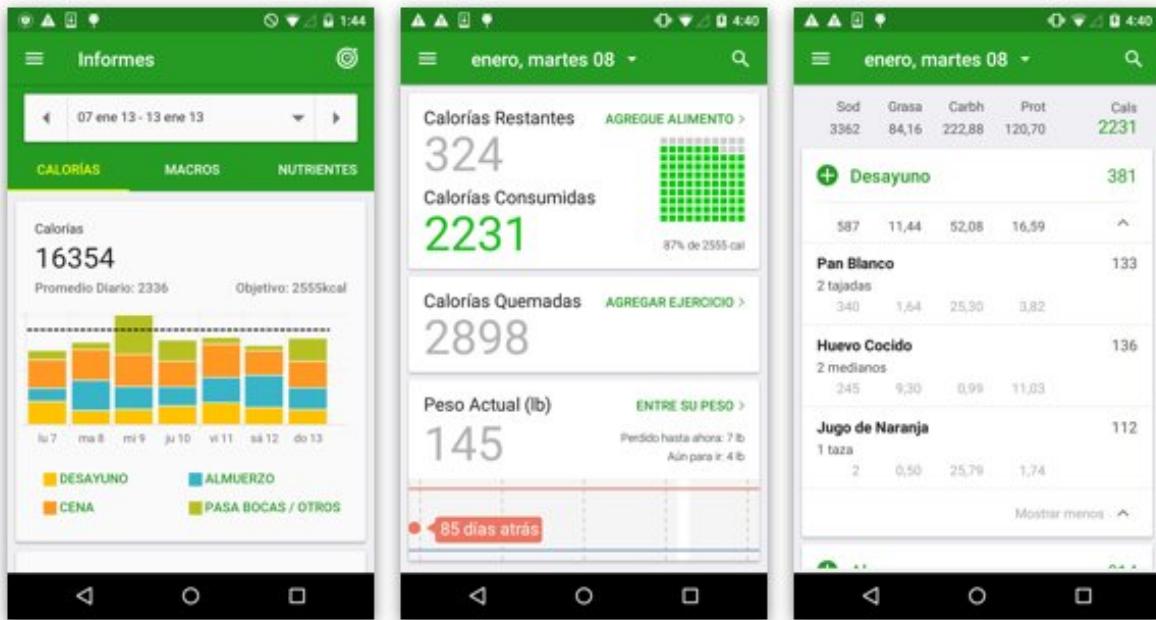


Figura 2.2: Diferentes vistas de la aplicación **Oorenji**

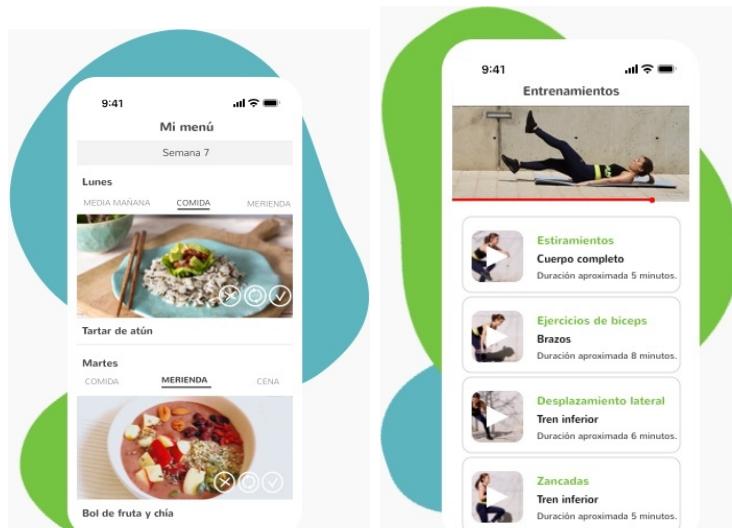
FatSecret

Es una aplicación [4] (ver figura 2.3) para Android que además de contabilizar las calorías ingeridas y quemadas, puede analizar el código de barras de los alimentos. Además, ayuda a controlar la cantidad de los tres macronutrientes (proteínas, carbohidratos y grasas) que se ingieren diariamente.

Figura 2.3: Distintas pantallas de la aplicación **FatSecret**

Nootric

Es una aplicación [5] (ver figura 2.4) en la que se pueden realizar diferentes planes nutricionales con recetas. Incluye recetas de cocina, guías y consejos para adelgazar y una lista de la compra por cada semana del mes. También dispone de una versión de pago, la cual cuenta con un mayor número de planes nutricionales así como un seguimiento a medida por parte de un nutricionista.



(a) Vista del menú

(b) Vista de entrenamientos

Figura 2.4: Distintas pantallas de la aplicación **Nootric**

Fitia

Es una aplicación [6] (ver figura 2.5) en la que se puede visualizar cada una de las comidas del día además de contar las calorías y calcular los nutrientes presentes en las recetas que ofrece. Dispone de varias opciones para alcanzar el objetivo deseado, desde aumentar masa muscular, reducir grasa corporal o simplemente mantener una dieta sana y saludable.

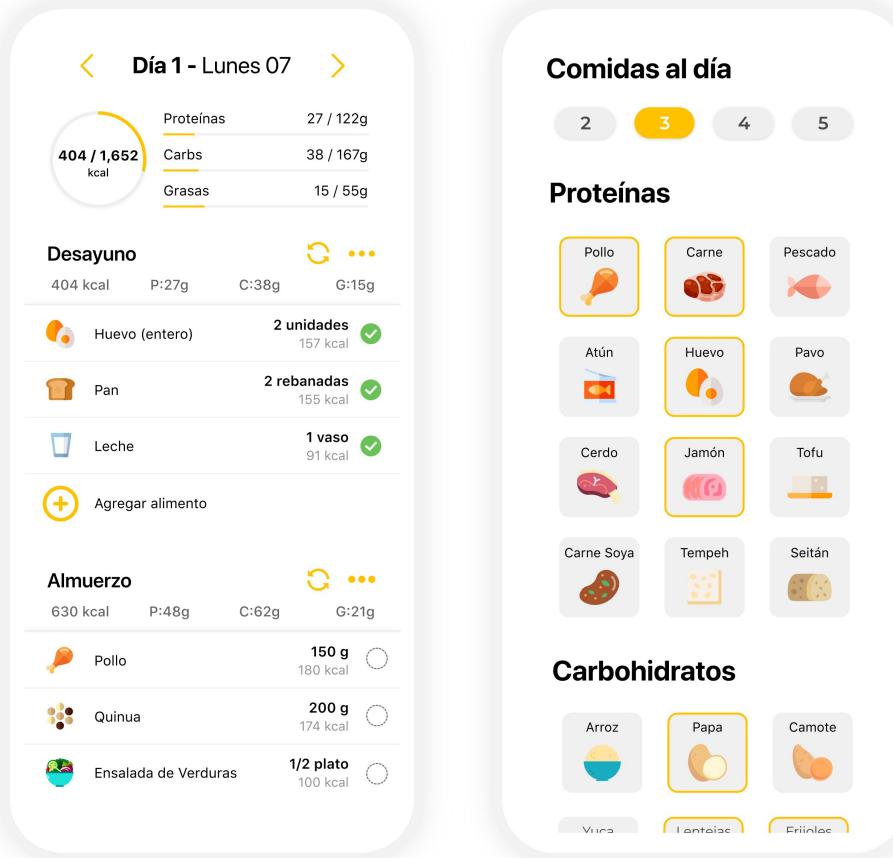


Figura 2.5: Distintas pantallas de la aplicación **Fitia**

Capítulo 3

Tecnología empleada

En este capítulo se van a comentar las diferentes tecnologías que se han empleado para la realización de este proyecto.

3.1. Herramientas utilizadas en la parte del cliente

3.1.1. Android Studio

Android Studio [7] es el entorno de desarrollo integrado oficial para la plataforma Android. Fue el entorno que reemplazó a Eclipse para el desarrollo de aplicaciones Android. Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicada de forma gratuita.

Como ocurre con la mayoría de entornos de desarrollo modernos, ofrece las herramientas necesarias para la generación del código, lo que se denomina la lógica de la aplicación. Estos entornos también ofrecen los mecanismos con los que se puede diseñar la interfaz de usuario que lucirá el desarrollo final.

Este tipo de entornos ofrecen además la posibilidad de ejecutar la aplicación en un emulador Android directamente desde el IDE. Es una herramienta potente ya que se puede visualizar la aplicación en este emulador sin la necesidad de compilar la aplicación e instalarla en un dispositivo físico, además de poder elegir la marca y tamaño de la pantalla del propio dispositivo.

Toda la parte visual de la aplicación ha sido desarrollada en Android Studio, utilizando la herramienta de diseño de vistas donde se pueden arrastrar y colocar en la posición deseada los diferentes elementos que la componen.

3.2. Herramientas utilizadas en la parte del servidor

3.2.1. Google Firebase

Firebase [8] es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles adquirida por Google en 2014. Es una plataforma ubicada en la nube, integrada con Google Cloud Platform, que usa un conjunto de herramientas para la creación y sincronización de proyectos.

Algunas ventajas de usar esta plataforma son las siguientes:

- Sincroniza los datos de los proyectos fácilmente sin necesidad de administrar conexiones o escribir lógica de sincronización.
- Utiliza un conjunto de herramientas multiplataforma: se puede integrar tanto en plataformas web como en aplicaciones móviles. A su vez, es compatible con grandes plataformas, entre las que destacan iOS y Android.
- Utiliza la infraestructura de Google y permite escalar cualquier tipo de aplicación, desde la más pequeña hasta la más potente.
- No es necesaria la creación de un servidor ya que las herramientas se incluyen en los SDK para los dispositivos móviles y web.

Authentication

La mayoría de las aplicaciones necesitan identificar a los usuarios. Para ello, se necesita poder guardar los datos en la nube de forma segura, proporcionando una misma experiencia en todos los dispositivos del usuario. Google Firebase proporciona las herramientas necesarias para ello.

Algunas ventajas de usar esta plataforma son las siguientes:

- Firebase Authentication [9] proporciona servicios de *backend* y bibliotecas de IU para hacer posible la autenticación de los usuarios. Además admite la autenticación por diversos medios, entre los que pueden destacar los proveedores de identidad federadas más populares, como Google o Twitter.
- Firebase aplica la misma administración de las contraseñas que utilizan otros servicios conocidos como Smart Lock o el administrador de contraseñas de Google.
- Firebase Authentication se integra con otros servicios de Firebase, y se aprovecha de estándares de la industria para poder integrar de forma sencilla dichos métodos con un backend personalizado.

En DietNow este servicio se ha utilizado para el registro de usuarios y para el sistema de gestión de permisos y seguridad.

Realtime Database

Firebase Realtime Database [10] es una base de datos NoSQL alojada en la nube que permite almacenar y sincronizar datos entre tus usuarios en tiempo real.

Firebase proporciona una base de datos en tiempo real, organizada en forma de árbol JSON. El servicio proporciona una API que garantiza la sincronización y almacenamiento de la información en la nube. Además, habilita la integración de dicho servicio con distintas aplicaciones realizadas en diferentes entornos, entre los que pueden destacar Android o iOS.

A su vez, gracias a la sincronización en tiempo real de la base de datos, permite a los usuarios acceder a ella desde cualquier dispositivo en tiempo real, obteniendo los datos actualizados.

Además Realtime Database cuenta con una funcionalidad interesante, si un usuario realiza cambios y pierde la conexión a Internet, el SDK de la plataforma usa una caché local en el dispositivo donde guarda dichos cambios. Una vez restablecida la conexión a Internet, se sincronizan los datos locales de forma automática.

En DietNow este servicio se ha utilizado para el almacenamiento de la información generada en la aplicación.

Storage

Cloud Storage [11] se diseñó para ayudar a almacenar y procesar con rapidez y facilidad el contenido generado por usuarios, como fotos y documentos.

El SDK de Firebase para Cloud Storage se integra en Firebase Authentication para proporcionar un control de acceso intuitivo y sencillo. Cuenta además con un modelo de seguridad declarativa, con el fin de permitir el acceso a los distintos archivos, dependiendo de la identidad del usuario o de las distintas propiedades del archivo, como el nombre, el tamaño, tipo de contenido y otros meta datos.

Firebase Storage proporciona cargas y descargas seguras de archivos para aplicaciones Firebase, sin importar la calidad de la red. Se puede utilizar para almacenar imágenes, audio, vídeo, o cualquier otro contenido generado por el usuario. Firebase Storage se basa en el almacenamiento de Google Cloud Storage.

En DietNow este servicio se ha utilizado para la carga, descarga y almacenamiento de archivos con extensión .pdf que se utilizan en la aplicación, así como las fotos de perfil de los usuarios.

3.2.2. Spring Boot

Spring Boot [12] es un *framework* para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Está dividido en múltiples módulos los cuales se pueden combinar para crear un proyecto de varios módulos. Entre todos los módulos disponibles se pueden destacar *Spring Web* o *Spring Security*, básicos y esenciales para la creación y desarrollo de aplicaciones web.

Durante el desarrollo de esta aplicación, se ha utilizado Spring Boot para crear una API a la cual los administradores hacen peticiones para crear nuevos usuarios y modificar tanto su correo electrónico o contraseña en el módulo de Google Firebase Authentication, como se ha explicado anteriormente en la sección 3.2.1. Para evitar que cualquier persona pueda realizar peticiones para crear o alterar información de los usuarios, se han empleado una serie de *tokens* -válidos durante el día- únicos para el administrador que realiza la petición con el objetivo de garantizar la legitimidad de la acción.

3.2.3. Node.js

Ideado como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js [13] está diseñado para crear aplicaciones *network* escalables. Está pensado para ser más liviano y eficiente que las aplicaciones en tiempo real de uso de datos que se ejecutan en los dispositivos.

La motivación de usar Node.js en la aplicación ha sido la de realizar una API propia que formatease la respuesta de la llamada a la API de Open Food Facts, debido a que la implementación de esta última en un entorno Android [14] suponía añadir un número elevado de clases a las ya existentes, lo cuál dificultaba la comprensión y el desarrollo del resto de código de la aplicación.

3.2.4. Retrofit

Retrofit [15] es un cliente o librería de Android que facilita la incorporación de las peticiones REST a las diferentes APIs expuestas en internet. Permite realizar todos los tipos de peticiones existentes GET, PUT, POST, DELETE, PATCH y HEAD.

Además, se encarga de serializar los objetos automáticamente, de modo que solo es necesario utilizar el objeto porque Retrofit lo transforma, obteniéndolo de la API REST. Los manejadores de peticiones son almacenados en interfaces Java a los que solo se les debe indicar el tipo de petición, endpoint de acceso y el objeto POJO de respuesta.

En la implementación de la aplicación, se ha utilizado Retrofit para llamar a la API en Spring para gestionar los usuarios con la API de Admin Firebase Authentication y para llamar a la API de Node.js para obtener información nutricional de los productos a través de Open Food Facts.

3.3. Otras herramientas

3.3.1. Git

Git [16] es un software de control de versiones de código abierto y gratuito. Inicialmente fue planeado para trabajar con varios desarrolladores en el núcleo de Linux. Se trata de un rastreador de contenido que se usa principalmente para almacenar código. Git posee un sistema de control de versiones para que varios desarrolladores puedan trabajar en paralelo sobre la misma aplicación permitiéndoles revertir y regresar a una versión anterior de su código.

En DietNow, este servicio se ha utilizado para trabajar en un repositorio donde almacenar las distintas versiones de la aplicación, así como para poder trabajar de forma conjunta en el proyecto.

3.3.2. Trello

Trello [17] es un software de gestión en línea basado en la metodología Kanban, que permite a los usuarios trabajar de forma colaborativa, utilizando tarjetas de trabajo en un tablero, llevando de este modo una “línea de producción” de tareas con sus estatus correspondientes.

Este software ha sido utilizado para la gestión del proyecto DietNow, más específicamente para la organización de las tareas de desarrollo a realizar, la asignación de cada tarea a los distintos miembros del equipo de desarrollo y el conocimiento en todo momento del progreso de cada tarea.

3.3.3. Overleaf

Overleaf [18] es una herramienta de publicación y redacción colaborativa en línea que hace más eficiente el proceso de redacción, edición y publicación de documentos.

Overleaf ofrece un editor L^AT_EX fácil de usar, con posibilidad de colaboración en tiempo real y una vista previa cargada automáticamente en segundo plano a medida que escribe.

Ha sido utilizado, junto con el libro B^AS_PX [19] de L^AT_EX, para la creación de la documentación relativa al proyecto DietNow.

3.3.4. Google Drive

Google Drive [20] es un servicio de Google de almacenamiento de archivos en la nube.

Este servicio ha sido utilizado para almacenar toda la documentación relativa al proyecto de DietNow.

3.3.5. Visual Studio Code

Visual Studio Code [21] es un editor de código gratuito y de código abierto desarrollado por Microsoft. Posee soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Es altamente personalizable ya que admite la instalación de distintas extensiones, cambios de temas, atajos de teclado y/o preferencias.

En DietNow este editor ha sido utilizado para el desarrollo de las APIs en Spring y Node.js necesarias para acceder a la base de datos de alimentos de Open Food Facts y para desarrollar otras funcionalidades mediante la API de Google Firebase.

Capítulo 4

Casos de uso

En este cuarto capítulo se listan los distintos actores que componen el sistema. A continuación se muestran los diferentes casos de uso agrupados por cada uno de esos actores.

4.1. Actores del sistema

Los actores del sistema son:

Deportistas

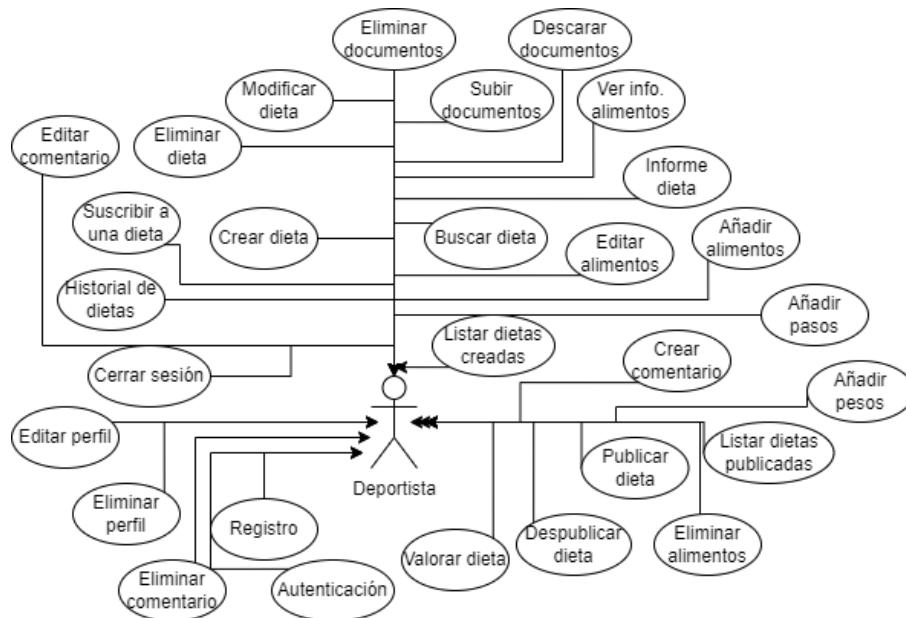
Se trata del actor principal y usuario final de la aplicación. Cualquier usuario que se registre tendrá asignado de forma automática este rol y podrá realizar las diferentes actividades asociadas.

Administradores

Se trata de los actores encargados de supervisar el correcto funcionamiento y de la gestión de la información introducida en la aplicación. Los usuarios con este rol serán capaces de modificar cualquier dieta creada, eliminar un deportista de la aplicación, crear dietas, etc.

4.2. Módulo deportista

En esta sección se describen los diferentes casos de uso asociados a este actor (ver figura 4.1). En dicha figura se pueden observar los diferentes casos de uso asociados al deportista.

Figura 4.1: Diagrama de casos de uso asociado al actor *Deportista*

| CU-01 | Registrar deportista |
|------------------------|--|
| Descripción | El deportista podrá registrarse en la aplicación para poder utilizar las funciones básicas de la misma |
| Actor | Deportista |
| Entrada | Datos del deportista |
| Salida | Mensaje de éxito o error |
| Origen | Página de registro |
| Precondición | No existe un usuario con el mismo correo electrónico |
| Postcondición si éxito | El deportista se registra correctamente |
| Postcondición si fallo | Se vuelven a pedir los datos para proceder nuevamente al registro |
| Flujo normal | <ol style="list-style-type: none"> 1. Se pulsa en el botón de registro 2. Se introducen los datos requeridos 3. Se validan los datos 4. Se pulsa sobre el botón de registrar y se da de alta el deportista |
| Flujo alternativo | <ol style="list-style-type: none"> 2. Se introducen de forma incorrecta los datos o lo hace con los de otro usuario ya registrado 3. Se muestra un mensaje de error y se vuelven a pedir los datos |

Tabla 4.1: Registrar deportista

| CU-02 | Autenticar deportista |
|------------------------|---|
| Descripción | El deportista puede iniciar sesión en la aplicación con sus credenciales |
| Actor | Deportista |
| Entrada | Credenciales de la cuenta del deportista |
| Salida | Mensaje de éxito o error |
| Origen | Página de inicio de sesión de la aplicación |
| Precondición | El deportista debe tener una cuenta creada con las credenciales proporcionadas |
| Postcondición si éxito | El deportista inicia sesión satisfactoriamente |
| Postcondición si fallo | Se vuelven a pedir los datos nuevamente |
| Flujo normal | <ol style="list-style-type: none"> 1. Se pulsa en el botón de iniciar sesión 2. Se introducen las credenciales necesarias para la acción 3. Se validan las credenciales y se inicia sesión |
| Flujo alternativo | <ol style="list-style-type: none"> 2. Se introducen unas credenciales incorrectas 3. Los datos no son validados y se muestra un mensaje de error |

Tabla 4.2: Autenticar deportista

| CU-03 | Cerrar la sesión de un deportista |
|------------------------|---|
| Descripción | El deportista cierra su sesión activa en la aplicación |
| Actor | Deportista |
| Entrada | NA |
| Salida | NA |
| Origen | Desplegable de opciones del perfil del deportista |
| Precondición | El deportista debe de haber iniciado sesión |
| Postcondición si éxito | Se cierra la sesión activa del deportista |
| Postcondición si fallo | La sesión del deportista no se cierra debido a un error |
| Flujo normal | <ol style="list-style-type: none"> 1. Se accede desde el menú principal al perfil del deportista 2. Se pulsa sobre las opciones del perfil 3. Se selecciona la opción de cerrar sesión |
| Flujo alternativo | NA |

Tabla 4.3: Cerrar la sesión de un deportista

| CU-04 | Modificar datos de un deportista |
|------------------------|--|
| Descripción | El deportista modifica los datos de su cuenta |
| Actor | Deportista |
| Entrada | Los datos modificados que se van a actualizar |
| Salida | Mensaje de éxito o error |
| Origen | Vista de modificar datos del deportista |
| Precondición | El deportista debe de haber iniciado sesión |
| Postcondición si éxito | Los datos del deportista son actualizados en la aplicación |
| Postcondición si fallo | Se vuelven a pedir los datos nuevamente tras haber mostrado un mensaje de error |
| Flujo normal | <ol style="list-style-type: none"> 1. Se accede desde el menú principal al perfil del deportista 2. Se pulsa sobre las opciones del perfil 3. Se selecciona la opción de editar perfil 4. Se introducen los nuevos datos que se quieran modificar 5. Se validan y actualizan en la aplicación |
| Flujo alternativo | <ol style="list-style-type: none"> 4. El deportista introduce algún dato erróneo o vacío y se muestra un mensaje de error |

Tabla 4.4: Modificar datos de un deportista

| CU-05 | Eliminar perfil de un deportista |
|------------------------|--|
| Descripción | Se elimina la cuenta del deportista |
| Actor | Deportista |
| Entrada | Confirmación del deportista |
| Salida | Mensaje de éxito o error si se ha borrado el perfil satisfactoriamente |
| Origen | Vista del perfil del deportista |
| Precondición | El deportista debe tener una cuenta activa y debe de haber iniciado sesión |
| Postcondición si éxito | Se elimina la cuenta del deportista del sistema |
| Postcondición si fallo | La cuenta del deportista no se ve alterada |
| Flujo normal | <ol style="list-style-type: none"> 1. Se accede desde el menú principal al perfil del deportista 2. Se pulsa sobre las opciones del perfil 3. Se selecciona la opción de eliminar cuenta 4. Se elimina la cuenta del deportista tras la confirmación |
| Flujo alternativo | <ol style="list-style-type: none"> 4. El deportista deniega la confirmación de borrar cuenta y no sucede nada |

Tabla 4.5: Eliminar perfil de un deportista

| CU-06 | Creación de una dieta |
|------------------------|--|
| Descripción | El deportista crea una dieta en el sistema |
| Actor | Deportista |
| Entrada | Datos de la dieta |
| Salida | Mensaje de éxito o error |
| Origen | Vista de crear una dieta |
| Precondición | El deportista debe tener una cuenta activa y debe de haber iniciado sesión |
| Postcondición si éxito | Se crea la dieta para el deportista |
| Postcondición si fallo | Mensaje de error y se vuelven a pedir los datos |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre las dietas creadas por el deportista 2. Se pulsa sobre el botón inferior en la zona derecha con el símbolo “+“ 3. Se introducen los datos requeridos para crear la dieta 4. Se pulsa sobre el botón de guardar y se crea la dieta |
| Flujo alternativo | <ol style="list-style-type: none"> 3. El deportista no rellena algún campo requerido 4. Se pulsa sobre el botón guardar y aparece un mensaje de error |

Tabla 4.6: El deportista crea una dieta

| CU-07 | Subir documentos a una dieta |
|------------------------|--|
| Descripción | El deportista sube documentos a una de sus dietas creadas |
| Actor | Deportista |
| Entrada | Documento seleccionado de los archivos del smartphone |
| Salida | Mensaje de éxito o error |
| Origen | Vista de editar dieta creada por el deportista |
| Precondición | La dieta debe pertenecer al deportista |
| Postcondición si éxito | El documento es asociado a la dieta |
| Postcondición si fallo | El documento no se sube y por lo tanto el estado de la dieta es alterado |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas creadas 2. Se accede a la dieta a la que se quiera subir el documento 3. Se pulsa sobre el botón de editar dieta 4. El deportista pulsa sobre el botón de subir documento 5. Se selecciona un documento del smartphone |
| Flujo alternativo | <ol style="list-style-type: none"> 5. El deportista no selecciona ningún documento y la operación de subida se cancela |

Tabla 4.7: Subir documentos a una dieta

| CU-08 | Eliminar documentos de una dieta |
|------------------------|--|
| Descripción | El deportista elimina documentos a una de sus dietas creadas |
| Actor | Deportista |
| Entrada | Identificador del documento que se quiere eliminar |
| Salida | Mensaje de éxito o error |
| Origen | Vista de edición de la dieta |
| Precondición | El documento debe existir y estar asociado a la dieta |
| Postcondición si éxito | El documento es eliminado de la dieta |
| Postcondición si fallo | El documento no se elimina y por lo tanto sigue asociado a la dieta |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas creadas 2. Se accede a la dieta de la que se quiera eliminar un documento 3. Se pulsa sobre el botón de editar la dieta 4. Se pulsa sobre la “X” situada junto al documento |
| Flujo alternativo | NA |

Tabla 4.8: Eliminar documentos de una dieta

| CU-09 | Descargar documentos de una dieta |
|------------------------|---|
| Descripción | El deportista descarga un documento de una dieta |
| Actor | Deportista |
| Entrada | Identificador del documento que se quiere descargar |
| Salida | NA |
| Origen | Vista detallada de una dieta |
| Precondición | El documento debe existir y estar asociado a la dieta |
| Postcondición si éxito | El documento es descargado en el smartphone del usuario |
| Postcondición si fallo | El documento no se descarga debido a un error |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas publicadas 2. Se accede a la dieta de la que se quiera descargar el documento 3. Se pulsa sobre el ícono de la flecha para descargar el documento |
| Flujo alternativo | NA |

Tabla 4.9: Descargar documentos de una dieta

| CU-10 | Suscribirse a una dieta |
|------------------------|--|
| Descripción | El deportista se suscribe a una de las dietas publicadas en el sistema |
| Actor | Deportista |
| Entrada | Confirmación del deportista |
| Salida | Mensaje de éxito o error |
| Origen | Vista de todas las dietas publicadas del sistema |
| Precondición | La dieta a la que se quiera suscribir debe estar publicada |
| Postcondición si éxito | El deportista se suscribe a la dieta elegida |
| Postcondición si fallo | NA |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de dietas publicadas 2. Se accede a ver la dieta que desea seguir 3. Se pulsa sobre el ícono “★” para empezar a seguir la dieta |
| Flujo alternativo | <ol style="list-style-type: none"> 4. Si el deportista ya sigue una dieta se muestra un aviso 5. Se confirma que quiere seguir la nueva dieta y se suscribe a dicha dieta |

Tabla 4.10: Suscribirse a una dieta

| CU-11 | Modificar una dieta creada por el deportista |
|------------------------|--|
| Descripción | El deportista modifica los diferentes campos de una de sus dietas creadas o añade alimentos a la misma |
| Actor | Deportista |
| Entrada | Datos nuevos de la dieta |
| Salida | Mensaje de éxito o error |
| Origen | Vista de las dietas creadas por el deportista |
| Precondición | La dieta que se quiere editar debe existir y pertenecer al deportista |
| Postcondición si éxito | Los datos de la dieta se modifican satisfactoriamente |
| Postcondición si fallo | Aviso del error sucedido |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre las dietas creadas 2. Se accede a ver la dieta que se quiera editar 3. Se pulsa sobre el botón de editar la dieta 4. Se actualizan los campos de la dieta o se añaden, eliminan y/o modifican alimentos de la misma 5. Se pulsa sobre el botón de guardar para actualizar la dieta |
| Flujo alternativo | <ol style="list-style-type: none"> 4. El deportista introduce de forma incorrecta alguno de los campos de la dieta 5. Se muestra un mensaje de error mostrando el fallo ocurrido |

Tabla 4.11: Modificar una dieta creada por el deportista

| CU-12 | Ver historial de dietas seguidas |
|------------------------|--|
| Descripción | El deportista puede ver el histórico de dietas seguidas anteriormente |
| Actor | Deportista |
| Entrada | NA |
| Salida | Listado de todas las dietas que ha seguido el deportista |
| Origen | Vista del histórico de dietas |
| Precondición | El deportista debe existir en el sistema |
| Postcondición si éxito | Se muestra una tabla con información de las dietas que ha seguido el deportista |
| Postcondición si fallo | NA |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de ver perfil 2. Se pulsa sobre las opciones del perfil 3. Se pulsa sobre ver el historial de dietas 4. Se muestran todas las dietas que seguido el deportista |
| Flujo alternativo | NA |

Tabla 4.12: Ver historial de dietas seguidas

| CU-13 | Eliminar una dieta creada por del deportista |
|------------------------|---|
| Descripción | El deportista elimina una de las dietas que ha creado |
| Actor | Deportista |
| Entrada | Confirmación del deportista |
| Salida | Mensaje de éxito o error |
| Origen | Vista de ver detalles de la dieta |
| Precondición | La dieta que se quiera eliminar debe existir |
| Postcondición si éxito | Mensaje de éxito y eliminación de la dieta |
| Postcondición si fallo | Mensaje de error indicando el fallo |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas creadas 2. Se pulsa sobre ver dieta para acceder a la vista en detalle de la misma 3. Se pulsa sobre el botón de eliminar dieta 4. Se muestra y acepta un mensaje de confirmación para eliminarla |
| Flujo alternativo | 4. El deportista deniega la confirmación de eliminar la dieta y no se realiza ninguna acción sobre la misma |

Tabla 4.13: Eliminar una dieta creada por del deportista

| CU-14 | Listar dietas creadas por del deportista |
|------------------------|--|
| Descripción | Se muestran todas las dietas que el deportista ha creado |
| Actor | Deportista |
| Entrada | Identificador del deportista |
| Salida | Todas las dietas del deportista |
| Origen | Vista de ver todas las dietas |
| Precondición | El deportista debe tener una cuenta activa en el sistema |
| Postcondición si éxito | Se muestran todas las dietas que el deportista ha creado |
| Postcondición si fallo | NA |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas creadas 2. Todas las dietas creadas por el deportista son listadas |
| Flujo alternativo | NA |

Tabla 4.14: Listar dietas creadas por del deportista

| CU-15 | Ver todas las dietas publicadas del sistema |
|------------------------|--|
| Descripción | Se muestran todas las dietas publicadas del sistema |
| Actor | Deportista |
| Entrada | NA |
| Salida | Todas las dietas publicadas que hay en el sistema |
| Origen | Vista de todas las dietas publicadas |
| Precondición | Tiene que haber dietas que los usuarios hayan publicado |
| Postcondición si éxito | Se muestran todas las dietas publicadas en formato tabla |
| Postcondición si fallo | NA |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas publicadas 2. Todas las dietas publicadas en el sistema son listadas |
| Flujo alternativo | NA |

Tabla 4.15: Ver todas las dietas publicadas del sistema

| CU-16 | Buscar dietas |
|------------------------|---|
| Descripción | El deportista busca una dieta entre todas las dietas publicadas |
| Actor | Deportista |
| Entrada | Texto introducido por el deportista sobre el que buscar dietas |
| Salida | Todas las dietas publicadas cuyos títulos coinciden de cierta forma con el texto introducido |
| Origen | Vista de todas las dietas publicadas |
| Precondición | Tiene que haber dietas publicadas en el sistema |
| Postcondición si éxito | Se muestran todas las dietas publicadas con un título similar al texto introducido |
| Postcondición si fallo | No se muestra ninguna dieta en la tabla de la correspondiente vista |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas publicadas 2. Se escribe el texto deseado sobre la barra superior de búsqueda 3. Se listan todas las dietas cuyos títulos coinciden con el texto buscado |
| Flujo alternativo | <ol style="list-style-type: none"> 3. No hay dietas que coincidan con el texto de la búsqueda y por lo tanto ninguna dieta es listada |

Tabla 4.16: Buscar dietas

| CU-17 | Publicar una dieta creada por el deportista |
|------------------------|---|
| Descripción | El deportista publica una de sus dietas creadas |
| Actor | Deportista |
| Entrada | Identificador de la dieta que se quiere publicar |
| Salida | Mensaje de éxito o fallo si la dieta se ha publicado correctamente o no |
| Origen | Vista de ver la dieta |
| Precondición | La dieta debe existir y no debe estar publicada |
| Postcondición si éxito | Mensaje informando de que la dieta ha sido publicada correctamente |
| Postcondición si fallo | Mensaje informando del error que se ha producido |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas creadas 2. Se accede a ver la dieta que el deportista quiera publicar 3. Se pulsa sobre el botón de publicar dieta 4. Se muestra un mensaje informativo de que la dieta ha sido publicada |
| Flujo alternativo | <ol style="list-style-type: none"> 3. Sucede algún error al momento de publicar la dieta se muestra un mensaje y no se produce ninguna acción sobre la dieta |

Tabla 4.17: Publicar una dieta creada por el deportista

| CU-18 | Despublicar una dieta creada por el deportista |
|------------------------|--|
| Descripción | El deportista despublica una de sus dietas creadas |
| Actor | Deportista |
| Entrada | Identificador de la dieta que se quiere despublicar |
| Salida | Mensaje de éxito o fallo si la dieta se ha despublicado correctamente o no |
| Origen | Vista de ver la dieta |
| Precondición | La dieta debe existir y debe estar publicada |
| Postcondición si éxito | Mensaje informando de que la dieta ha sido despublicada correctamente |
| Postcondición si fallo | Mensaje informando del error que se ha producido |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas creadas 2. Se accede a ver la dieta que el deportista quiera despublicar 3. Se pulsa sobre el botón de despublicar dieta 4. Se muestra un mensaje informativo de que la dieta ha sido despublicada |
| Flujo alternativo | 3. Sucede algún error al momento de despublicar la dieta se muestra un mensaje y no se produce ninguna acción sobre la dieta |

Tabla 4.18: Despublicar una dieta creada por el deportista

| CU-19 | Añadir alimentos a una dieta |
|------------------------|---|
| Descripción | El deportista añade alimentos a una de sus dietas creadas |
| Actor | Deportista |
| Entrada | Información del alimento que se quiere añadir |
| Salida | Mensaje de éxito o error |
| Origen | Vista de editar dieta creada por el deportista |
| Precondición | La dieta debe pertenecer al deportista |
| Postcondición si éxito | El alimento es añadido a la dieta |
| Postcondición si fallo | El alimento no se añade y el estado de la dieta no se ve alterado |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas creadas 2. Se accede a la dieta a la que se quiera añadir el alimento 3. Se pulsa sobre el ícono "+" situado bajo la descripción 4. En este punto el deportista puede escoger entre dos opciones <ol style="list-style-type: none"> 4.1. Añadir mediante la cámara escaneando el código de barras 4.2. Añadir manualmente, bien especificando el código de barras o añadiendo la información requerida para crear un alimento |
| Flujo alternativo | 4. El deportista cancela la operación y no se añade el alimento |

Tabla 4.19: Añadir alimentos a una dieta

| CU-20 | Eliminar alimentos de una dieta |
|------------------------|--|
| Descripción | El deportista elimina alimentos de sus dietas creadas |
| Actor | Deportista |
| Entrada | Identificador del alimento que se quiere eliminar |
| Salida | Mensaje de éxito o error |
| Origen | Vista de editar dieta creada por el deportista |
| Precondición | El alimento debe estar entre los añadidos a la dieta |
| Postcondición si éxito | El alimento es eliminado de la dieta |
| Postcondición si fallo | El alimento no se elimina y el estado de la dieta no se ve alterado |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas creadas 2. Se accede a la dieta a la que se quiera eliminar el alimento 3. Se pulsa sobre el botón de editar la dieta 4. Se pulsa sobre el ícono de la papelera situado junto al alimento |
| Flujo alternativo | 4. El deportista decide no borrar el alimento y el estado de la dieta no se ve alterado |

Tabla 4.20: Eliminar alimentos de una dieta

| CU-21 | Editar alimentos de una dieta |
|------------------------|---|
| Descripción | El deportista edita la información de un alimento de una dieta |
| Actor | Deportista |
| Entrada | Nuevos datos del alimento que se va a editar |
| Salida | Mensaje de éxito o error |
| Origen | Vista de editar dieta creada por el deportista |
| Precondición | El alimento debe estar entre los añadidos a la dieta |
| Postcondición si éxito | El alimento de la dieta es editado |
| Postcondición si fallo | El alimento no se edita y el estado de la dieta no se ve alterado |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas creadas 2. Se accede a la dieta a la que se quiera eliminar el alimento 3. Se pulsa sobre el botón de editar la dieta 4. Se cambian los campos del alimentos que se quieran editar 5. Se pulsa sobre el ícono de guardado situado junto al alimento |
| Flujo alternativo | 5. El deportista no pulsa sobre el botón de guardar y por lo tanto el alimento no es editado |

Tabla 4.21: Editar alimentos de una dieta

| CU-22 | Ver información nutricional de los alimentos a una dieta |
|------------------------|--|
| Descripción | El deportista consulta la información nutricional de un alimento |
| Actor | Deportista |
| Entrada | Identificador del alimento del cual se quiere consultar la información |
| Salida | Alerta emergente con la información nutricional del alimento |
| Origen | Vista de información detallada de la dieta |
| Precondición | El alimento debe estar entre los añadidos a la dieta |
| Postcondición si éxito | La información del alimento es mostrada al deportista |
| Postcondición si fallo | La información del alimento no se muestra |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver todas las dietas publicadas o creadas por el propio deportista 2. Se accede a la dieta a la que se quiera información de alimentos 3. Se pulsa sobre el botón del ojo situado junto al alimento 4. Se muestra una alerta con la información detallada del alimento |
| Flujo alternativo | NA |

Tabla 4.22: Ver información nutricional de los alimentos a una dieta

| CU-23 | Informe gráfico de la dieta seguida |
|------------------------|---|
| Descripción | El deportista consulta gráficamente la evolución de las kcal consumidas a lo largo del tiempo |
| Actor | Deportista |
| Entrada | Alimentos consumidos durante los días que se está realizando la dieta |
| Salida | Gráfica informativa |
| Origen | Perfil del deportista |
| Precondición | El deportista debe de seguir una dieta y debe de haber consumido alimentos de dicha dieta |
| Postcondición si éxito | Informe gráfico de las kcal consumidas prolongado en el tiempo |
| Postcondición si fallo | La gráfica se muestra vacía sin datos |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver perfil 2. Se accede pulsa sobre el botón de ver informe gráfico de la dieta seguida |
| Flujo alternativo | NA |

Tabla 4.23: Informe gráfico de la dieta seguida

| CU-24 | Añadir comentarios a la dieta seguida |
|------------------------|--|
| Descripción | El deportista añade un comentario a la dieta que está siguiendo |
| Actor | Deportista |
| Entrada | Texto que se quiera comentar sobre la dieta |
| Salida | Mensaje de éxito o error |
| Origen | Vista detallada de comentarios de la dieta seguida |
| Precondición | El deportista debe de estar siguiendo una dieta |
| Postcondición si éxito | El comentario es publicado correctamente |
| Postcondición si fallo | El comentario no se publica en la dieta seguida |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de dieta seguida 2. Se pulsa sobre el botón de comentarios 3. Se introduce el texto que se quiera comentar y se pulsa sobre el botón de comentar |
| Flujo alternativo | <ol style="list-style-type: none"> 3. Tras introducir el texto el deportista no pulsa sobre comentar y por lo tanto el comentario no se publica |

Tabla 4.24: Añadir comentarios a la dieta seguida

| CU-25 | Editar comentarios a la dieta seguida |
|------------------------|--|
| Descripción | El deportista edita uno de sus comentarios publicados en la dieta seguida |
| Actor | Deportista |
| Entrada | Nuevo texto del comentario publicado |
| Salida | Mensaje de éxito o error |
| Origen | Vista detallada de comentarios de la dieta seguida |
| Precondición | El deportista debe de estar siguiendo una dieta y debe de haber comentado al menos una vez |
| Postcondición si éxito | El comentario es actualizado correctamente |
| Postcondición si fallo | El comentario no se actualiza |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de dieta seguida 2. Se pulsa sobre el botón de comentarios 3. En el área del comentario que quiera actualizar se introduce el nuevo texto 4. Se pulsa sobre el botón de actualizar situado bajo el comentario |
| Flujo alternativo | <ol style="list-style-type: none"> 4. El deportista no pulsa sobre el botón de actualizar y por lo tanto el comentario no se actualiza |

Tabla 4.25: Editar comentarios a la dieta seguida

| CU-26 | Eliminar comentarios a la dieta seguida |
|------------------------|--|
| Descripción | El deportista elimina uno de sus comentarios publicados en la dieta seguida |
| Actor | Deportista |
| Entrada | Identificador del comentario que se quiere borrar |
| Salida | Mensaje de éxito o error |
| Origen | Vista detallada de comentarios de la dieta seguida |
| Precondición | El deportista debe de estar siguiendo una dieta y debe de haber comentado al menos una vez |
| Postcondición si éxito | El comentario es eliminado correctamente |
| Postcondición si fallo | El comentario no se elimina y permanece sin alteraciones |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de dieta seguida 2. Se pulsa sobre el botón de comentarios 3. Se pulsa sobre el botón de borrar comentario situado bajo el mismo y se confirma la acción |
| Flujo alternativo | <ol style="list-style-type: none"> 3. El deportista no pulsa sobre el botón de borrar y por lo tanto el comentario no se elimina |

Tabla 4.26: Eliminar comentarios a la dieta seguida

| CU-27 | Añadir pasos caminados diariamente |
|------------------------|---|
| Descripción | El deportista añade los pasos que ha caminado durante el día |
| Actor | Deportista |
| Entrada | Número de pasos caminados |
| Salida | Mensaje de éxito o error |
| Origen | Vista del perfil del deportista |
| Precondición | El deportista debe de tener una cuenta activa |
| Postcondición si éxito | Los pasos son añadidos al sistema |
| Postcondición si fallo | El número de pasos no se añade |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver su perfil 2. Se pulsa sobre el botón con el icono “+“ 3. Se selecciona la opción de añadir pasos 4. Se introduce el número de pasos caminados durante el día especificado |
| Flujo alternativo | 4. El deportista no añade pasos y por lo tanto no se añaden al sistema |

Tabla 4.27: Añadir pasos caminados diariamente

| CU-28 | Añadir peso diario |
|------------------------|--|
| Descripción | El deportista añade su peso diario |
| Actor | Deportista |
| Entrada | Número de kilogramos totales del deportista |
| Salida | Mensaje de éxito o error |
| Origen | Vista del perfil del deportista |
| Precondición | El deportista debe de tener una cuenta activa |
| Postcondición si éxito | El peso es añadido al sistema |
| Postcondición si fallo | El número de kilogramos no se añade |
| Flujo normal | <ol style="list-style-type: none"> 1. El deportista pulsa sobre el botón de ver su perfil 2. Se pulsa sobre el botón con el icono “+“ 3. Se selecciona la opción de añadir peso 4. Se introduce el número de kilogramos del deportista |
| Flujo alternativo | 4. El deportista no añade peso y por lo tanto no se añade al sistema |

Tabla 4.28: Añadir peso diario

| CU-29 | Actualizar información de la dieta actual |
|------------------------|--|
| Descripción | Se actualiza información de la dieta actual de los días de la semana |
| Actor | Deportista |
| Entrada | Identificador de la dieta seguida |
| Salida | Información de la dieta actual |
| Origen | Vista de la dieta seguida |
| Precondición | Se ha iniciado sesión con una cuenta con el rol de deportista y debe de seguir una dieta |
| Postcondición si éxito | Se actualiza la información de la dieta actual |
| Postcondición si fallo | No se actualizan los alimentos consumidos durante la semana |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de dieta seguida 2. En el día actual, se introducen la cantidad (gramos) consumida de cada alimento 3. Se marca el alimento que se quiere actualizar 4. Se pulsa sobre el botón de guardar |
| Flujo alternativo | <ol style="list-style-type: none"> 3. No se selecciona ningún alimento y no se actualiza la información de la dieta 4. No se pulsa en el botón de guardar y la información de la dieta no se altera |

Tabla 4.29: El deportista actualiza la información de la dieta seguida

| CU-30 | Valorar la dieta seguida |
|------------------------|---|
| Descripción | El deportista valora la dieta seguida |
| Actor | Deportista |
| Entrada | Identificador de la dieta seguida |
| Salida | Mensaje de confirmación |
| Origen | Vista de la dieta seguida |
| Precondición | Se ha iniciado sesión con una cuenta con el rol de deportista y debe de seguir una dieta |
| Postcondición si éxito | Se actualiza la valoración de dieta |
| Postcondición si fallo | La valoración de la dieta seguida no cambia |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de dieta seguida 2. Se pulsa el botón de me gusta o no me gusta en función de la valoración que se quiera dar |
| Flujo alternativo | NA |

Tabla 4.30: El deportista valora la dieta seguida

| CU-31 | Dejar de seguir la dieta |
|------------------------|---|
| Descripción | El deportista deja de seguir la dieta |
| Actor | Deportista |
| Entrada | Identificador de la dieta seguida |
| Salida | Mensaje de confirmación |
| Origen | Vista de la dieta seguida |
| Precondición | Se ha iniciado sesión con una cuenta con el rol de deportista y debe de seguir una dieta |
| Postcondición si éxito | Se deja de seguir la dieta |
| Postcondición si fallo | No se deja de seguir la dieta |
| Flujo normal | 1. Desde el menú principal se pulsa sobre el botón de dieta seguida 2. Se pulsa el botón de dejar de seguir la dieta |
| Flujo alternativo | NA |

Tabla 4.31: El deportista deja de seguir la dieta

4.3. Módulo administrador

En esta sección se muestran los casos de uso asociados al actor con el rol de administrador (ver figura 4.2).

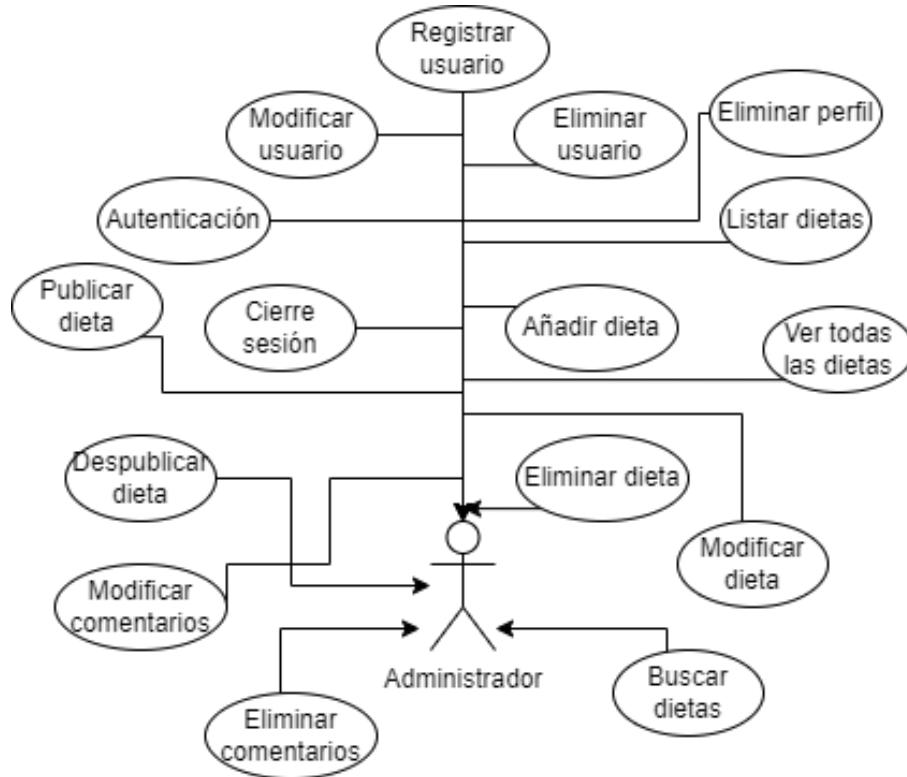


Figura 4.2: Diagrama de casos de uso asociado al actor *Administrador*

| CU-32 | Registrar manualmente un usuario |
|------------------------|---|
| Descripción | El administrador crea un usuario en el sistema con el rol de deportista administrador |
| Actor | Administrador |
| Entrada | Datos del usuario que se va a crear |
| Salida | Mensaje de éxito o error |
| Origen | Vista de crear usuarios del administrador |
| Precondición | Tener una cuenta activa con el rol administrador |
| Postcondición si éxito | El usuario se crea correctamente |
| Postcondición si fallo | Se vuelven a pedir los datos del nuevo usuario en el caso de haberse producido un fallo |
| Flujo normal | 1. El administrador hace click en el botón de crear usuario desde su página principal |
| | 2. Se introducen los datos requeridos para crear dicho usuario |
| | 3. El usuario se crea satisfactoriamente con las credenciales proporcionadas |
| Flujo alternativo | 2. Se introduce algún dato clave ya existente, como el correo electrónico |
| | 3. Se muestra un mensaje de error y se vuelven a pedir los datos |

Tabla 4.32: Registrar manualmente un usuario

| CU-33 | Modificar los datos de un usuario |
|------------------------|--|
| Descripción | El administrador modifica los datos de un usuario del sistema |
| Actor | Administrador |
| Entrada | Nuevo datos del usuario que se va a crear |
| Salida | Mensaje de éxito o error |
| Origen | Vista de modificar el perfil de un usuario |
| Precondición | El usuario que se quiere modificar debe existir en el sistema |
| Postcondición si éxito | Los datos del usuario son actualizados correctamente |
| Postcondición si fallo | Se vuelven a pedir los nuevos datos del usuario |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador hace click en el botón de todos los usuarios desde su página principal 2. Se accede al perfil del usuario que se quiera modificar 3. Se editan los campos que sean necesarios 4. Se pulsa sobre el botón de guardar |
| Flujo alternativo | 3. Se introduce algún dato clave ya existente, como el correo electrónico, y se muestra un mensaje de error |

Tabla 4.33: Modificar los datos de un usuario

| CU-34 | Eliminar a un usuario del sistema |
|------------------------|--|
| Descripción | El administrador desactiva la cuenta de un usuario |
| Actor | Administrador |
| Entrada | Identificador del usuario que se quiera desactivar |
| Salida | Mensaje de éxito o error |
| Origen | Vista de modificar el perfil de un usuario |
| Precondición | El usuario que se quiere modificar debe existir en el sistema |
| Postcondición si éxito | Los datos del usuario son actualizados correctamente |
| Postcondición si fallo | Se vuelven a pedir los nuevos datos del usuario |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador hace click en el botón de todos los usuarios desde su página principal 2. Se accede al perfil del usuario que se quiera modificar 3. Se editan los campos que sean necesarios 4. Se pulsa sobre el botón de guardar |
| Flujo alternativo | 3. Se introduce algún dato clave ya existente, como el correo electrónico, y se muestra un mensaje de error |

Tabla 4.34: Eliminar a un usuario del sistema

| CU-35 | Autenticar administrador |
|------------------------|--|
| Descripción | Etapa de inicio de sesión en la aplicación con el rol de administrador. |
| Actor | Administrador |
| Entrada | Datos necesarios para el inicio de sesión |
| Salida | Mensaje de error o éxito |
| Origen | Página de inicio de sesión |
| Precondición | Tener una cuenta creada y activa con ese correo |
| Postcondición si éxito | El administrador ha iniciado sesión correctamente |
| Postcondición si fallo | Se vuelven a pedir los datos de inicio de sesión |
| Flujo normal | <ol style="list-style-type: none"> 1. Se pulsa sobre el botón de login. 2. Se introducen los datos de inicio de sesión 3. Se comprueban los datos y si son correctos se inicia sesión |
| Flujo alternativo | 3. Los datos introducidos no son correctos y se vuelve a la página de inicio de sesión |

Tabla 4.35: Autenticar administrador

| CU-36 | Cerrar la sesión de un administrador |
|------------------------|--|
| Descripción | El administrador cierra su sesión activa en la aplicación |
| Actor | Administrador |
| Entrada | NA |
| Salida | NA |
| Origen | Vista del perfil del administrador |
| Precondición | El administrador debe existir y debe tener la sesión iniciada |
| Postcondición si éxito | Se cierra la sesión del administrador |
| Postcondición si fallo | La sesión del administrador no se ve alterada |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador pulsa en el botón de ver perfil 2. Se pulsa sobre las acciones del perfil en la parte superior derecha 3. Se selecciona la opción de cerrar sesión |
| Flujo alternativo | NA |

Tabla 4.36: Cerrar la sesión de un administrador

| CU-37 | Añadir nueva dieta al sistema |
|------------------------|---|
| Descripción | El administrador añade una nueva dieta al sistema |
| Actor | Administrador |
| Entrada | Datos requeridos para crear una dieta |
| Salida | Mensaje de éxito o error |
| Origen | Vista de crear una dieta |
| Precondición | El administrador debe tener una cuenta activa en el sistema |
| Postcondición si éxito | La dieta se crea correctamente para ese usuario |
| Postcondición si fallo | Se vuelven a pedir los datos de la dieta tras haber mostrado un mensaje informando del fallo |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador pulsa sobre el botón de ver sus dietas creadas 2. Se pulsa sobre el botón “+“ en la parte inferior derecha 3. Se introducen los datos requeridos de la dieta a crear 4. Se pulsa sobre el botón de crear dieta |
| Flujo alternativo | <ol style="list-style-type: none"> 3. El administrador no completa los campos requeridos para crear una dieta 4. Se muestra un mensaje de error y se vuelven a pedir los datos |

Tabla 4.37: El administrador crea una nueva dieta

| CU-38 | Eliminar una dieta del sistema |
|------------------------|--|
| Descripción | Eliminar una dieta del sistema |
| Actor | Administrador |
| Entrada | Identificador de la dieta que se quiere eliminar |
| Salida | Mensaje de error o éxito |
| Origen | Vista de la dieta que quieras eliminar |
| Precondición | Se ha iniciado sesión con una cuenta con el rol de administrador y existe la dieta |
| Postcondición si éxito | Dieta eliminada correctamente |
| Postcondición si fallo | Mensaje de error |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de todas las dietas 2. Se accede a la vista detalle de la dieta que quieras eliminar 3. Se pulsa sobre el botón de eliminar |
| Flujo alternativo | 4. El usuario no acepta la confirmación de eliminar la dieta |

Tabla 4.38: Eliminar una dieta del sistema

| CU-39 | Modificar una dieta del sistema |
|------------------------|---|
| Descripción | Se modifican algunos de los datos de una dieta |
| Actor | Administrador |
| Entrada | Datos de la dieta que el administrador quiere modificar |
| Salida | Mensaje de error o éxito |
| Origen | Vista de la dieta que quieras modificar |
| Precondición | Se ha iniciado sesión con una cuenta con el rol de administrador y existe la dieta |
| Postcondición si éxito | Dieta modificada con los nuevos datos introducidos previamente |
| Postcondición si fallo | La dieta no ha sido modificada por lo que no se cambia ningún dato |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de todas las dietas 2. Se accede a la vista detalle de la dieta que quieras modificar 3. Se pulsa sobre el botón de modificar. 4. Se introducen los datos que se quieren modificar de la dieta 5. Se pulsa sobre el botón de guardar |
| Flujo alternativo | 5. No se pulsa sobre el botón de guardar |

Tabla 4.39: Modificar una dieta del sistema

| CU-40 | Publicar una dieta del sistema |
|------------------------|--|
| Descripción | El administrador publica cualquier dieta sin publicar del sistema |
| Actor | Administrador |
| Entrada | Identificador de la dieta que se quiera publicar |
| Salida | Mensaje de éxito o error |
| Origen | Vista detalle de la dieta que se quiera publicar |
| Precondición | El administrador debe tener una cuenta activa y la dieta debe existir y estar despublicada |
| Postcondición si éxito | La dieta es publicada y visible para todos los usuarios |
| Postcondición si fallo | El estado de publicación de la dieta no se ve alterado |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador pulsa sobre el botón de ver todas las dietas del sistema 2. Se accede a la dieta que se quiera publicar 3. Se pulsa sobre el botón de publicar dieta |
| Flujo alternativo | 3. El administrador no pulsa ningún botón de cambiar el estado de la dieta |

Tabla 4.40: Publicar una dieta del sistema

| CU-41 | Despublicar una dieta del sistema |
|------------------------|--|
| Descripción | El administrador despublica cualquier dieta publicada del sistema |
| Actor | Administrador |
| Entrada | Identificador de la dieta que se quiera despublicar |
| Salida | Mensaje de éxito o error |
| Origen | Vista detalle de la dieta que se quiera despublicar |
| Precondición | El administrador debe tener una cuenta activa y la dieta debe existir y estar publicada |
| Postcondición si éxito | La dieta es despublicada y deja de ser visible para todos los usuarios |
| Postcondición si fallo | El estado de publicación de la dieta no se ve alterado |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador pulsa sobre el botón de ver todas las dietas del sistema o sobre el botón de dietas publicadas en el sistema 2. Se accede a la dieta que se quiera despublicar 3. Se pulsa sobre el botón de despublicar dieta |
| Flujo alternativo | 3. El administrador no pulsa ningún botón de cambiar el estado de la dieta |

Tabla 4.41: Despublicar una dieta del sistema

| CU-42 | Modificar comentarios de una dieta |
|------------------------|--|
| Descripción | Se modifican los comentarios de una dieta |
| Actor | Administrador |
| Entrada | Comentario que se quiere modificar sobre la dieta |
| Salida | Mensaje de éxito |
| Origen | Vista de todos los comentarios de la dieta seleccionada previamente |
| Precondición | Se ha iniciado sesión con una cuenta con el rol de administrador y existe la dieta |
| Postcondición si éxito | Dieta modificada con los comentarios modificados previamente |
| Postcondición si fallo | Los comentarios de la dieta no han sido modificados por lo que no se actualiza ningún comentario |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde el menú principal se pulsa sobre el botón de todas las dietas. 2. Se accede a la vista detalle de la dieta que quieres actualizar el comentario 3. Se pulsa sobre el botón de comentarios 4. Se actualiza el comentario de la dieta especificada 5. Se pulsa sobre el botón de actualizar |
| Flujo alternativo | 5. No se pulsa sobre el botón de actualizar |

Tabla 4.42: Modificar comentarios de una dieta

| CU-43 | Eliminar comentarios de una dieta |
|------------------------|---|
| Descripción | El administrador elimina un comentario de una dieta publicada |
| Actor | Administrador |
| Entrada | Identificador del comentario que se quiera eliminar |
| Salida | Mensaje de éxito o error |
| Origen | Vista detalle de los comentarios de la dieta publicada |
| Precondición | El administrador debe tener una cuenta activa y la dieta debe existir, estar publicada y tener al menos un comentario que eliminar |
| Postcondición si éxito | El comentario es borrado de la dieta publicada |
| Postcondición si fallo | El comentario no es eliminado |
| Flujo normal | <ol style="list-style-type: none"> 1. El administrador pulsa sobre el botón de ver todas las dietas publicadas 2. Se accede a la dieta que tiene el comentario que se quiera borrar 3. Se pulsa sobre el botón de comentarios 4. En las opciones del comentario se pulsa sobre eliminar |
| Flujo alternativo | 4. El administrador no pulsa ningún botón de las acciones del comentario |

Tabla 4.43: Eliminar comentarios de una dieta

| CU-44 | Eliminar perfil de un administrador |
|------------------------|--|
| Descripción | Se elimina la cuenta del administrador |
| Actor | Administrador |
| Entrada | Confirmación del usuario |
| Salida | Mensaje de éxito o error si se ha borrado el perfil satisfactoriamente |
| Origen | Vista del perfil del administrador |
| Precondición | El usuario debe tener una cuenta activa y debe de haber iniciado sesión |
| Postcondición si éxito | Se elimina la cuenta del usuario del sistema |
| Postcondición si fallo | La cuenta del administrador no se ve alterada |
| Flujo normal | <ol style="list-style-type: none"> 1. Se accede desde el menú principal al perfil del usuario 2. Se pulsa sobre las opciones del perfil 3. Se selecciona la opción de eliminar cuenta 4. Se elimina la cuenta del administrador tras la confirmación |
| Flujo alternativo | 4. El administrador deniega la confirmación de borrar cuenta y no sucede nada |

Tabla 4.44: Eliminar perfil de un administrador

| CU-45 | Listar dietas creadas por del administrador |
|------------------------|---|
| Descripción | Se muestran todas las dietas que el administrador ha creado |
| Actor | Administrador |
| Entrada | Identificador del administrador |
| Salida | Todas las dietas del administrador |
| Origen | Vista de ver todas las dietas |
| Precondición | El administrador debe tener una cuenta activa en el sistema |
| Postcondición si éxito | Se muestran todas las dietas que el administrador ha creado |
| Postcondición si fallo | NA |
| Flujo normal | <ol style="list-style-type: none"> 1. Desde la página principal se pulsa sobre el botón de dietas creadas 2. Todas las dietas creadas por el administrador son listadas |
| Flujo alternativo | NA |

Tabla 4.45: Listar dietas creadas por del administrador

| CU-46 | Ver todas las dietas del sistema |
|------------------------|--|
| Descripción | Se muestran todas las dietas del sistema |
| Actor | Administrador |
| Entrada | NA |
| Salida | Todas las dietas que hay en el sistema |
| Origen | Vista de todas las dietas publicadas |
| Precondición | Tiene que haber dietas que los usuarios hayan creado |
| Postcondición si éxito | Se muestran todas las dietas en formato tabla |
| Postcondición si fallo | NA |
| Flujo normal | 1. Desde la página principal se pulsa sobre el botón de todas las dietas 2. Todas las dietas en el sistema son listadas |
| Flujo alternativo | NA |

Tabla 4.46: Ver todas las dietas publicadas del sistema

| CU-47 | Buscar dietas |
|------------------------|--|
| Descripción | El administrador busca una dieta entre todas las dietas |
| Actor | Administrador |
| Entrada | Texto introducido por el administrador sobre el que buscar dietas |
| Salida | Todas las dietas cuyos títulos coinciden de cierta forma con el texto introducido |
| Origen | Vista de todas las dietas |
| Precondición | Tiene que haber dietas en el sistema |
| Postcondición si éxito | Se muestran todas las dietas con un título similar al texto introducido |
| Postcondición si fallo | No se muestra ninguna dieta en la tabla de la correspondiente vista |
| Flujo normal | 1. Desde la página principal se pulsa sobre el botón de ver todas las dietas 2. Se escribe el texto deseado sobre la barra superior de búsqueda 3. Se listan todas las dietas cuyos títulos coinciden con el texto buscado |
| Flujo alternativo | 3. No hay dietas que coincidan con el texto de la búsqueda y por lo tanto ninguna dieta es listada |

Tabla 4.47: Buscar dietas

Capítulo 5

Arquitectura

En este capítulo se describe la arquitectura del sistema utilizado para este proyecto.

5.1. Arquitectura de la aplicación

La arquitectura utilizada en esta aplicación se basa en un sistema cliente servidor donde el cliente, mediante conexión a internet, realiza peticiones al servidor y se queda a la espera de la respuesta para realizar las diferentes acciones solicitadas. En la figura 5.1 se puede ver una representación gráfica de este sistema.

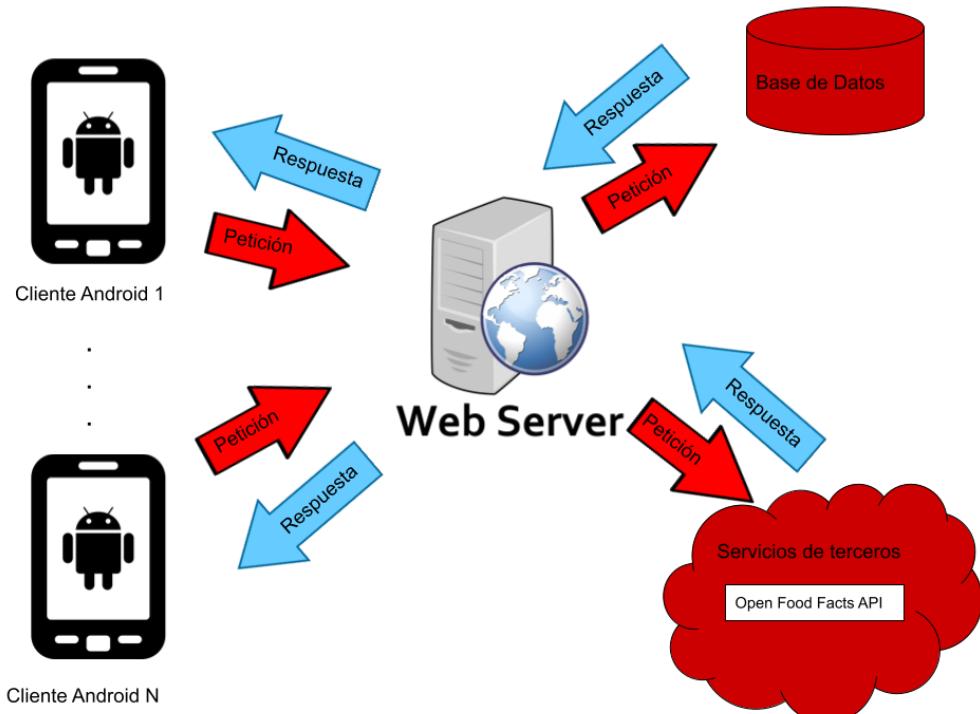


Figura 5.1: Esquema de la arquitectura de la aplicación

Como se puede observar en la figura 5.1, el usuario necesita tener su dispositivo conectado

a internet en todo momento para poder realizar peticiones al servidor web. El servidor a su vez se conecta con los diferentes proveedores de servicios, ya sea a la base de datos de la aplicación o a la API encargada de obtener los alimentos. En este proyecto el servidor web enviará y recibirá las peticiones de manera asíncrona para mejorar la eficiencia a la hora de comunicarse con los diferentes elementos del sistema.

5.2. Patrones de diseño y arquitectónicos

En este apartado se procederá a explicar los diferentes patrones de diseño y de arquitectura presentes en el proyecto.

5.2.1. Patrones arquitectónicos

Los patrones arquitectónicos [22] son aquellos que se encargan de dar una estructura esencial al proyecto con un nivel de abstracción mucho mayor a los patrones de diseño.

Modelo Vista Presentador (MVP)

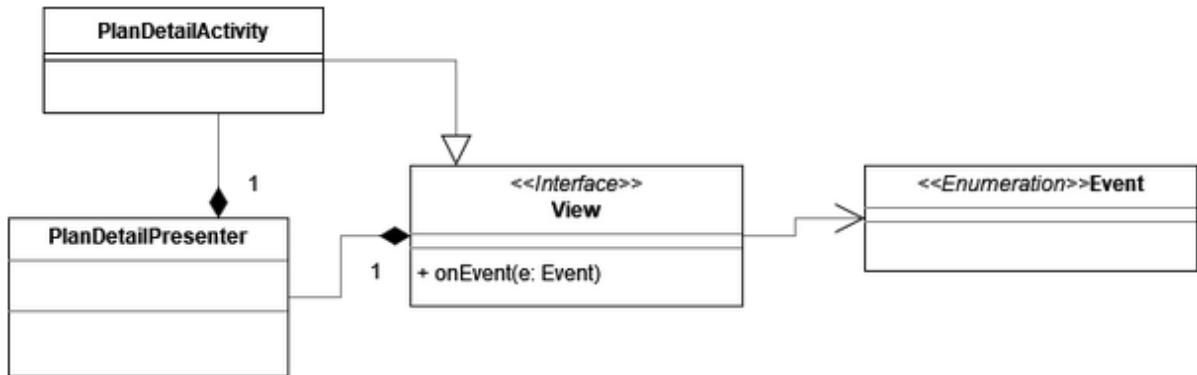


Figura 5.2: Esquema de Modelo Vista Presentador

El patrón MVP (Modelo-Vista-Presentador) es un patrón de arquitectura principalmente utilizado para generar interfaces de usuario. Como se puede apreciar en la figura 5.2, este patrón encapsula toda la lógica de la vista a través del presentador, recogiendo los datos del propio modelo del sistema. Es un diseño arquitectónico muy utilizado para la realización de aplicaciones Android o aplicaciones web desarrolladas mediante el lenguaje de programación ASP.NET. En modelo vista presentador, nos podemos encontrar tres diferentes elementos:

- **Modelo:** interfaz que define los datos y la estructura de los mismos.
- **Presentador:** clase mediadora entre la vista y el modelo. Es la encargada de actualizar los datos de la vista recogidas desde el modelo y actualizar el modelo con los datos recibidos de la vista.
- **Vista:** interfaz que muestra los datos obtenidos del presentador. Lo que el usuario ve al abrir la aplicación.

5.2.2. Patrones de diseño

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño.

Singleton

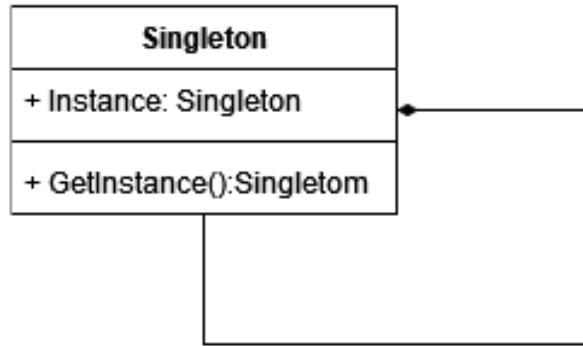


Figura 5.3: Esquema del patrón Singleton

El patrón Singleton (fig 5.3) es utilizado para garantizar una única instancia del objeto al que queremos acceder, haciendo el código más eficiente. Por cada referencia al uso de la clase, se proporciona un único punto de acceso a la instancia llamando al método `getInstance()`, el cual devuelve la instancia o la crea si antes no se ha llamado a dicho método.

Este patrón se ha usado en la aplicación Android para hacer única la capa de servicios de la aplicación encargada de la creación, principalmente, de los modelos correspondientes. La base de datos de Firebase también está creada mediante este patrón, teniendo una única instancia de referencia a la base de datos.

Façade

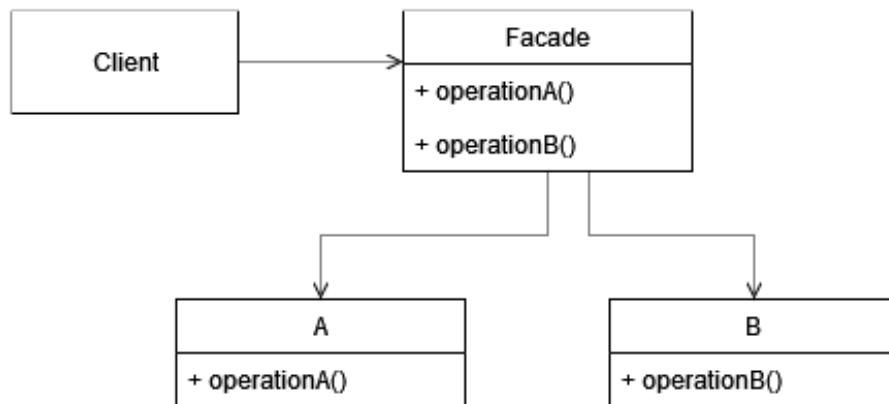


Figura 5.4: Esquema del patrón Façade

El Patrón Façade o patrón Fachada (fig 5.4) proporciona una interfaz unificada para un conjunto de interfaces del subsistema ocultando al cliente los componentes del subsistema,

reduciendo el número de objetos que accede el cliente. En este proyecto se ha utilizado este patrón para unificar las diferentes llamadas a las diferentes API generadas para la realización de este proyecto.

Adapter

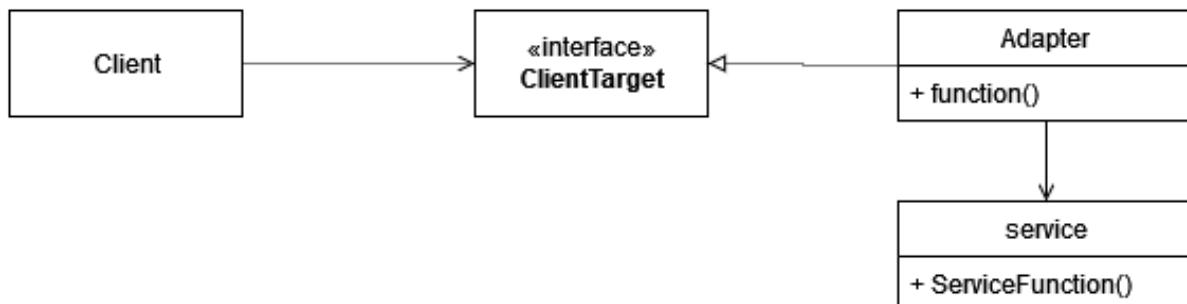


Figura 5.5: Esquema del patrón Adapter

El patrón Adaptador (fig 5.5) es el encargado de convertir la interfaz de un objeto para que otro objeto pueda comprenderla. Es por ello que se ha implementado este patrón para adaptar los productos obtenidos mediante la API haciéndolas un objeto entendible para el cliente. Otro de los motivos para los que se ha propuesto dicho patrón es para la creación de listas dinámicas en cada una de las vistas necesarias. Estas vistas se han desarrollado mediante el elemento RecyclerView propio de Android en el que mediante un adaptador se implementan las diferentes listas u objetos adaptados para que el cliente pueda observarlas con mayor claridad.

Transfer

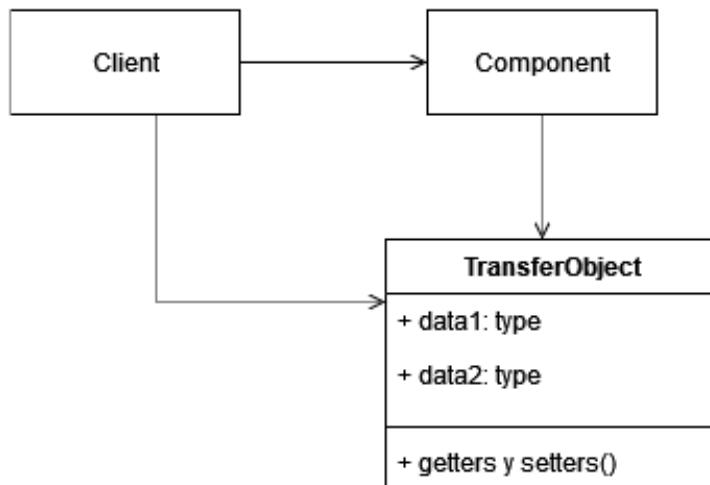


Figura 5.6: Esquema del patrón Transfer

El patrón Transfer (fig 5.6) es el encargado de agrupar varios datos en una misma clase facilitando el intercambio de información entre las diferentes capas del proyecto.

En este proyecto se ha utilizado el patrón para el propio propósito, para mostrar datos al usuario y para insertarlos a la propia base de datos.

View helper (ayudante de vista)

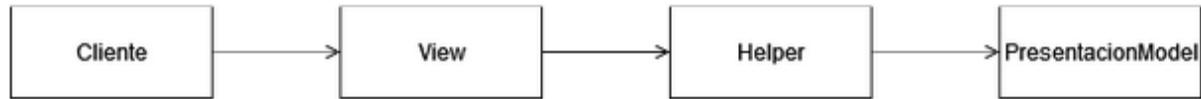


Figura 5.7: Esquema del patrón View Helper

El patrón View Helper o ayudante de vista (fig 5.7) se encarga de la creación de la vista mejorando la partición, reutilización y mantenibilidad de la misma separando los roles y facilitando la prueba. En este proyecto el patrón se utiliza con la utilización del RecyclerView de la plataforma Android, delegando toda la lógica de las listas dinámicas y su funcionamiento a la clase Adapter que necesita para generar la vista dinámica.

Capítulo 6

Modelo de datos

En este capítulo se describe el modelo de datos utilizado para persistir el sistema desarrollado.

6.1. Modelo E-R

Como se aprecia en la figura 6.1, el diagrama entidad relación representa las distintas entidades y atributos que componen el sistema. Toda la información almacenada se representa mediante las siguientes colección de documentos: Usuarios, Dietas, Alimentos, Comentarios, Pasos, Pesos e Historial de dietas.

El sistema guarda información de los usuarios, los cuales pueden crear dietas. A su vez, la dieta guarda información sobre la cantidad de cada alimento que tiene que ingerir el deportista. Para poder llevar a cabo la funcionalidad principal de la aplicación, el usuario puede crear dietas que posteriormente pueden ser seguidas por otros usuarios. Para realizar el seguimiento de la dieta del usuario, se crea un historial donde se almacena por cada día la cantidad consumida de cada alimento. El usuario además puede ver y publicar comentarios sobre la dieta que sigue actualmente.

La aplicación cuenta también con distintos usuarios administradores encargados de supervisar los diferentes usuarios y dietas del sistema.

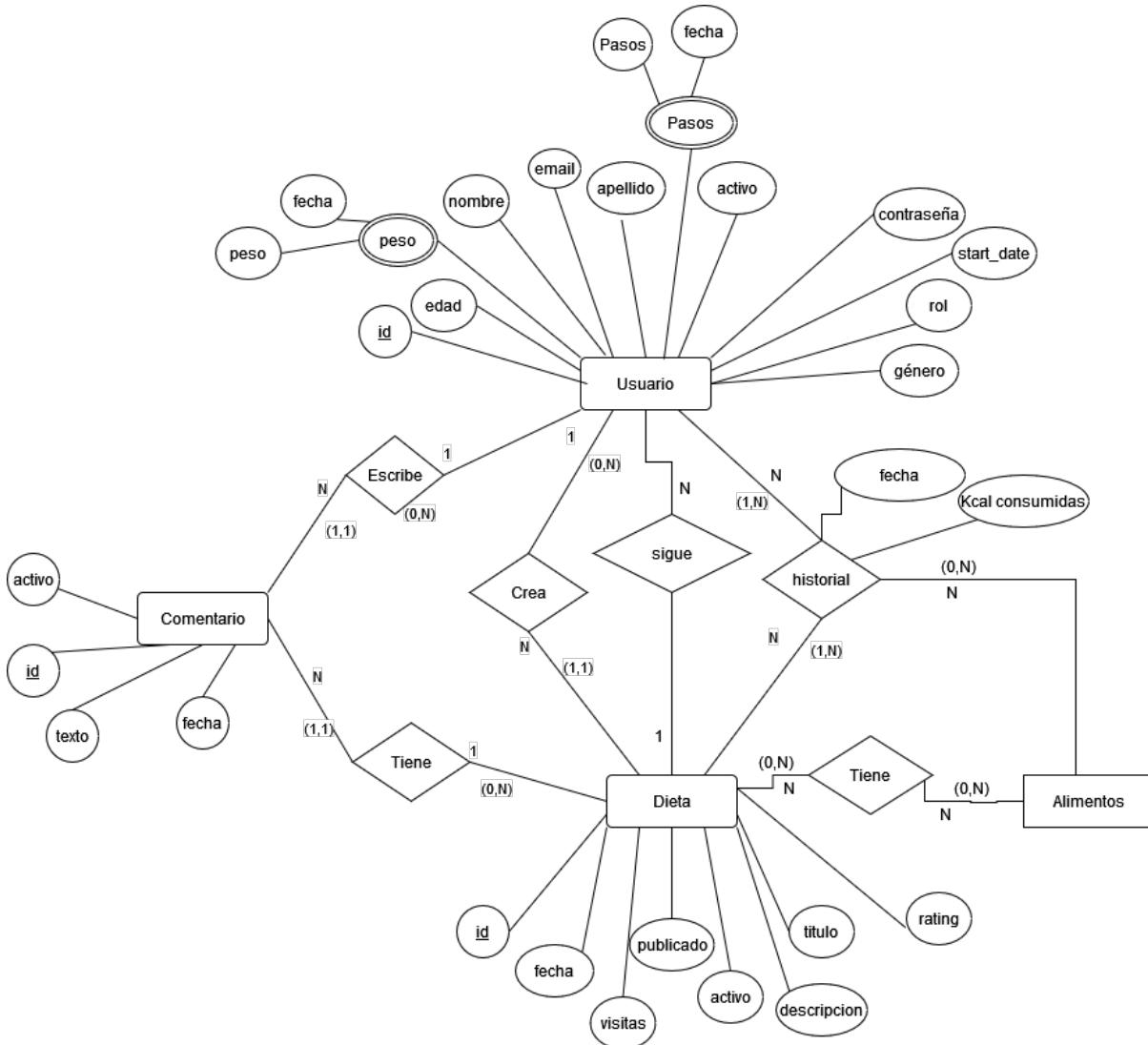


Figura 6.1: Diagrama Entidad Relación

6.2. Implementación de la base de datos

El modelo E-R se implementa mediante una base de datos de tipo NoSQL, donde los datos son almacenados como documentos JSON [23]. A continuación, se describen cada uno de ellos.

6.2.1. Colección de documentos de usuarios

Esta colección de documentos contiene la información de cada usuario registrado en la aplicación (ver figura 6.2).

Concretamente contiene una lista de identificadores únicos para cada usuario y para cada identificador están asociados los siguientes campos:

- **Identificador de usuario** generado automáticamente por Firebase.
 - **Active**: booleano que indica si la cuenta del usuario está activa o no. Todos los usuarios registrados tienen el valor **True** y los usuarios que se han dado de

baja tienen este campo con el valor a `False`.

- **Age:** campo numérico que indica la edad del usuario.
- **Diet:** contiene el ID de la dieta que está siguiendo actualmente el usuario.
- **Email:** almacena el correo electrónico del usuario, necesario para que éste pueda iniciar sesión en la aplicación, además de ser un canal de comunicación en caso de que sea necesario enviar una notificación al usuario.
- **Gender:** campo que almacena el género del usuario, los valores posibles son `MALE` (Hombre), `FEMALE` (Mujer), `NO_GENRE` (Prefiero no especificar).
- **Height:** campo que guarda la altura del usuario.
- **Lastname:** apellidos del usuario.
- **Name:** nombre del usuario.
- **Password:** campo que guarda encriptada la contraseña del usuario (mediante `jBcrypt` [24]). Cuando el usuario desea cambiar su contraseña, se comprueba este valor junto con la nueva contraseña cifrada para verificar que es distinta y proceder al cambio de la misma.
- **Role:** campo que almacena el rol del usuario, los valores posibles son `USER` y `ADMIN`.
- **Start date:** campo que guarda la fecha en la que se dio de alta el usuario en la aplicación.



Figura 6.2: Colección de documentos de los usuarios

6.2.2. Colección de documentos de dietas

Esta colección de documentos contiene la información de cada dieta registrado en la aplicación (ver figura 6.3).

Concretamente la colección de documentos de dietas contiene una lista de identificadores únicos para cada dieta registrada en la aplicación. Para cada identificador están asociados los siguientes campos:

- **Identificador de la dieta** generado automáticamente por Firebase.
- **Active**: booleano que indica si la dieta está activa o no. Todas las dietas creadas por defecto tienen el valor `True` y si el usuario autor de una dieta la ha dado de baja este campo tendrá el valor `False`.
- **Aliments**: campo que almacena una lista de alimentos, la estructura utilizada es una lista de identificadores de alimentos y dentro de cada identificador, la información del alimento en cuestión con los siguientes campos:
 - **Active**: booleano que indica si el alimento está activo o no.
 - **Grams**: los gramos que tiene el alimento.
 - **Kcal**: las kilocalorías del alimento.
 - **Name**: nombre del alimento.
- **Date**: la fecha en la que se creó la dieta.
- **Description**: campo que guarda la descripción que le ha puesto el autor a la dieta.
- **Id**: campo que almacena el identificador de la dieta
- **Published**: booleano que indica si la dieta está publicada o no. `True` indica que la dieta está publicada y la puede ver cualquier usuario de la aplicación y `False` indica que la dieta no está publicada y solo la puede ver su autor.
- **Rating**: campo que almacena una lista de usuarios que han valorado la aplicación. Guarda el id de los usuarios en un hashmap donde la clave es el id y el valor es un booleano, si el valor es `True` significa que el usuario ha dado *like* a la dieta, si el valor es `False` significa que el usuario ha dado *dislike*.
- **Title**: título de la dieta.
- **User**: campo que almacena el identificador del usuario que ha creado la dieta.
- **Visits**: campo que almacena una lista de usuarios que han visitado la aplicación. Guarda el id de los usuarios en un hashmap donde la clave es el id y el valor es un booleano a `True`.

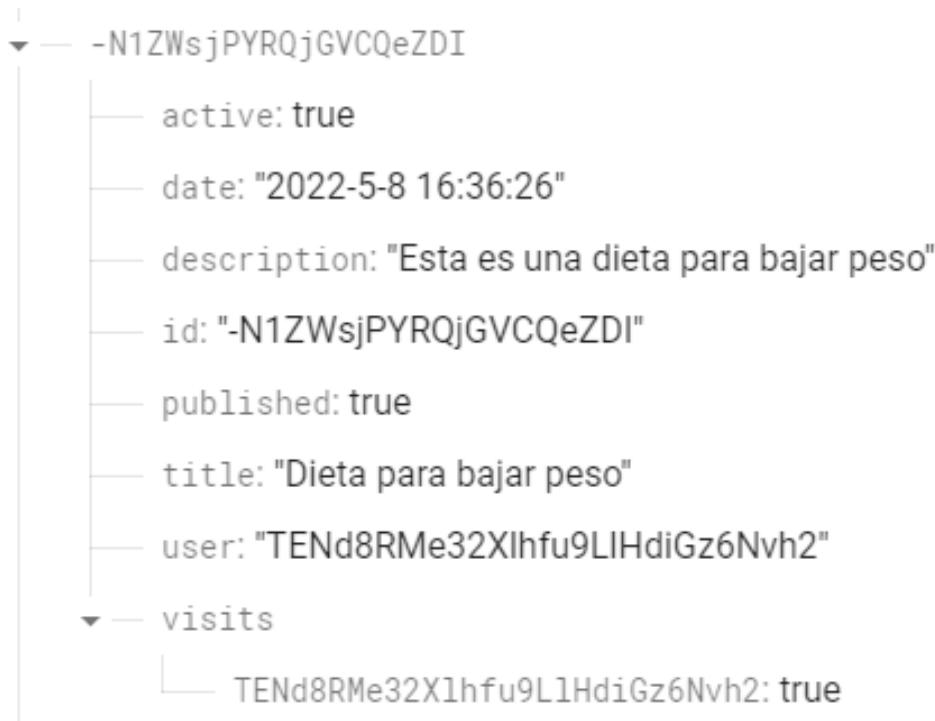


Figura 6.3: Entrada de la colección de documentos de dietas

6.2.3. Colección de documentos de alimentos

Esta colección de documentos contiene la información de cada alimento registrado en la aplicación (ver figura 6.4).

Concretamente contiene una lista de identificadores únicos. Existen 2 formatos de identificador para esta colección de documentos, los identificadores numéricos, el código de barras que identifica a cada alimento y los identificadores alfanuméricos que identifican a los alimentos que han sido dados de alta manualmente por el usuario. Independientemente del tipo de identificador, cada alimento registrado en la aplicación están asociados los siguientes campos:

- **Identificador del alimento** generado por la API Open Food Facts o manualmente.
 - **Active**: booleano que indica si el alimento está activo o no.
 - **Grams**: campo que almacena la cantidad de alimento que contiene una ración.
 - **Grams consumed**: campo que almacena la cantidad consumida del alimento. Este campo tiene utilidad cuando se calculan las kilocalorías consumidas en cada día de la semana de la dieta que se está siguiendo.
 - **Kcal**: la cantidad de kilocalorías que contiene una ración.
 - **Name**: nombre del alimento.

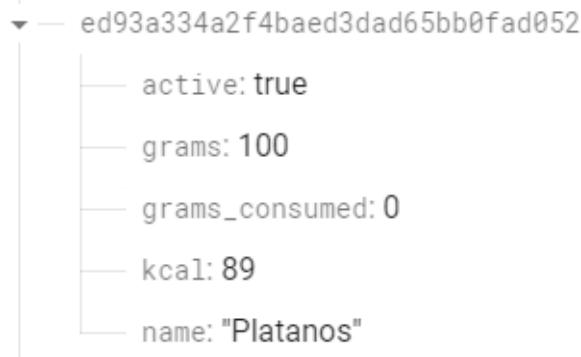


Figura 6.4: Entrada de la colección de documentos de alimentos

6.2.4. Colección de documentos de comentarios

Esta colección de documentos contiene la información de cada comentario registrado en la aplicación (ver figura 6.5).

Concretamente contiene una lista de identificadores de dietas que tienen al menos un comentario. Cada identificador de dieta, contiene dentro una lista de identificadores de comentarios. Dentro de cada identificador de comentario, se encuentran los siguientes campos:

- **Identificador de la dieta**
 - **Identificador del comentario** generado automáticamente por Firebase.
 - **Comment**: campo que contiene el comentario que ha escrito el usuario.
 - **Date**: la fecha y hora a la que se escribió el comentario.
 - **User**: el identificador del usuario que ha escrito el comentario.

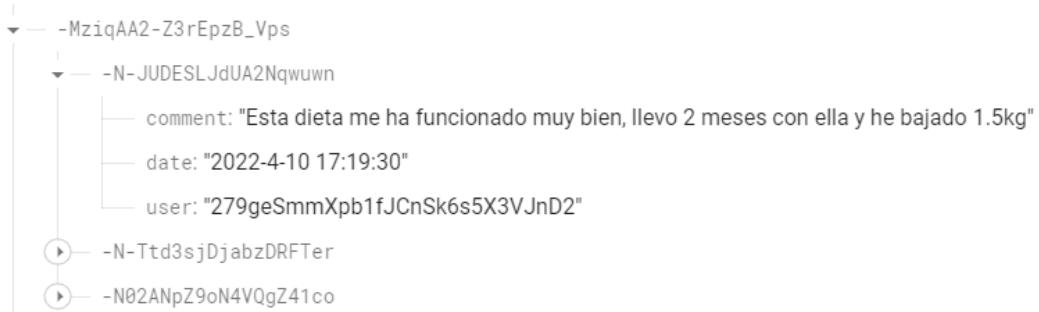


Figura 6.5: Entrada de la colección de documentos de comentarios

6.2.5. Colección de documentos de pasos

Esta colección de documentos contiene la información de los pasos realizados por cada usuario registrado en la aplicación (ver figura 6.6).

Concretamente contiene una lista de identificadores únicos de cada usuario. Cada identificador tiene asociado un hashmap de pares clave valor, donde la clave es la fecha y el valor son los pasos que ha hecho el usuario ese día. El propio funcionamiento de claves

de Firebase hace que las fechas insertadas se ordenen cronológicamente. La estructura de una entrada en la colección de documentos es la siguiente:

- **Identificador del usuario**

- **Fecha** en la que se registraron los pasos - **Nº de pasos** caminados durante ese día.

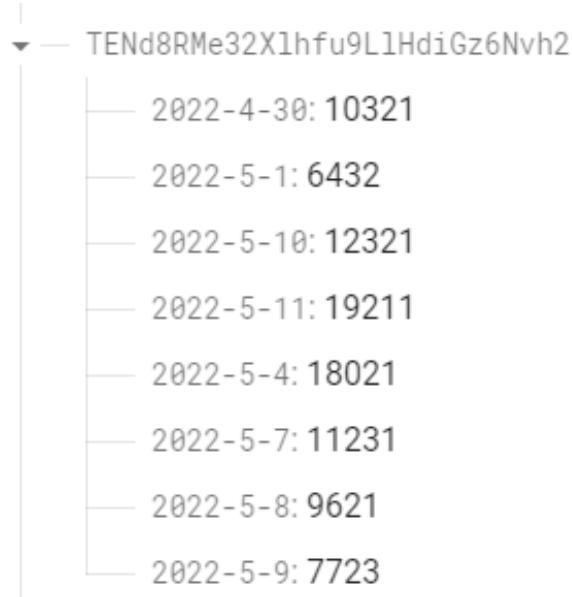


Figura 6.6: Entrada de la colección de documentos de pasos

6.2.6. Colección de documentos de pesos

Esta colección de documentos contiene la información del peso de cada usuario registrado en la aplicación (ver figura 6.7).

Concretamente contiene una lista de identificadores únicos de cada usuario. Cada identificador tiene asociado un hashmap de pares clave valor, donde la clave es la fecha y el valor es el peso que el usuario ha registrado ese día. El propio funcionamiento de claves de Firebase hace que las fechas insertadas se ordenen cronológicamente.

La estructura de una entrada en la colección de documentos es la siguiente:

- **Identificador del usuario**

- **Fecha** en la que se registró el peso - **Peso** del día indicado

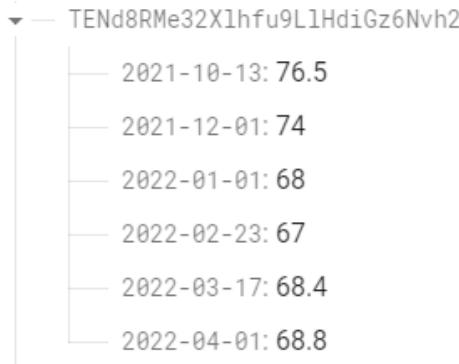


Figura 6.7: Entrada de la colección de documentos de pesos

6.2.7. Colección de documentos de historial de dietas

Esta colección de documentos contiene la información sobre las distintas dietas que ha seguido el usuario (ver figura 6.8).

La colección de documentos de historial de dietas contiene una lista de identificadores únicos de cada usuario. Cada uno de ellos tiene a su vez, una lista de los identificadores de al menos una las dietas que ha seguido. Cada dieta posee una entrada que indica la fecha en la que se consumió el alimento y, por cada una de estas fechas, se almacena la cantidad del alimento consumido.

- **Identificador del usuario**

- **Identificador de la dieta**
 - **Fecha** en la que se consumió el alimento
 - **Identificador del alimento - Cantidad de alimento consumido** en esa fecha



Figura 6.8: Entrada de la colección de documentos historial de dietas

6.3. Almacenamiento de archivos

Para el almacenamiento de archivos e imágenes se utiliza Firebase Storage, los datos que almacena son las imágenes de los usuarios y los documentos con extensión “.pdf” que se pueden añadir a una dieta.

La estructura que usa Firebase Storage es un sistema de directorios que tiene la siguiente forma:

- DietNow

- **diets**: carpeta donde se almacenan todos los documentos asociados a las dietas.
 - Identificador de la dieta: carpeta donde se almacenan los documentos de dicha dieta.
 - ◊ Nombre del documento pdf: hash MD5 del documento.
- **images**:
 - Imagen de perfil del usuario: para identificar cada imagen de usuario, éstas se suben con la estructura “**profile_<identificador del usuario>.jpg**“.

A continuación, en las figuras 6.9 y 6.10, se muestra una vista detallada en Firebase Storage de un documentos asociado a una dieta y la imagen de perfil de un usuario, respectivamente.

| Nombre | Tamaño | Tipo | Modificación más reciente |
|---|-----------|-----------------|---------------------------|
| 1ba2aeef6d81c79afe1dcc8e4bac7bb7c.pdf | 260.54 KB | application/pdf | 4 may 2022 |

Nombre: 1ba2aeef6d81c79afe1dcc8e4bac7bb7c.pdf...

Tamaño: 266,791 bytes

Tipo: application/pdf

Creado: 4 may 2022 20:56:04

Actualizado: 4 may 2022 20:56:04

Ubicación del archivo

Otros metadatos

Figura 6.9: Vista detallada de un documento de una dieta

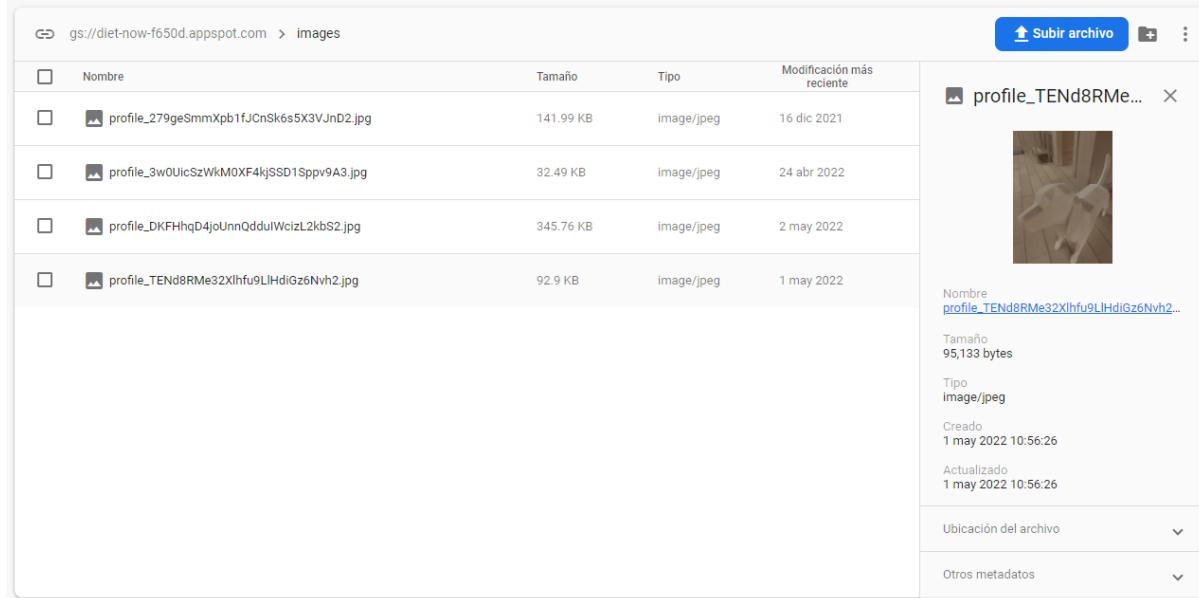


Figura 6.10: Vista detallada de la imagen de perfil de un usuario

Capítulo 7

Diseño de la aplicación

En este capítulo se describirá el diseño y la implementación de las funcionalidades de la aplicación. El objetivo final es conseguir una aplicación modular y escalable con un diseño simple e intuitivo para todos los roles de usuario.

7.1. Colores de la aplicación

Para el diseño de la aplicación, se ha escogido una gama de colores verdes como colores principales de la aplicación, y un color azul verdoso, para detalles más secundarios como el color inferior que se le añaden a los campos de texto, cuando se están rellenando por parte del usuario.

El motivo de la elección de esta gama de colores ha sido el de seguir una línea que se asemeje con el color de la fruta y las verduras, el verde por norma general. Además es el color que normalmente se suele asociar con la biología y lo biológico, ya que es el motivo principal de esta aplicación, el de seguir una serie de dietas sanas y saludables.

Otro de los motivos de la elección de estos colores, ha sido el contraste que hacen con el fondo blanco, sin llegar a resaltar demasiado ni ser llamativo, ya que para el diseño de la aplicación se buscó la línea más minimalista posible, sin imágenes de fondo ni colores calientes o llamativos.

7.2. Funcionalidad del cliente

La capa de cliente se ha implementado teniendo como público objetivo a los usuarios del sistema operativo Android.

El razonamiento detrás de esta decisión reside en los siguientes puntos:

- La cuota de mercado del sistema operativo Android a nivel europeo es aproximadamente del 70 %, de modo de la aplicación tiene una base de usuarios potenciales más amplia que cualquier otro sistema operativo para dispositivos móviles [25].
- Tiene código abierto y ofrece mayor libertad al poder desarrollar sin necesidad de pedir permisos.

- Android cuenta con un sistema multitarea que permite abrir distintas aplicaciones a la vez y hacerlas funcionar simultáneamente, así como ponerlas en modo suspensión, si no las estamos utilizando.
- El sistema de sincronización de Android en la nube demuestra una elevada eficiencia, ya que al estar ligado a Google, los servicios web cuentan con un gran sistema de trabajo.
- La aplicación al estar instalada en el dispositivo, se ejecuta mucho más rápido que si tuviera que cargar e interpretar código proveniente de otra fuente.

7.2.1. Ciclo de vida de una actividad

Todas las actividades o pantallas visibles de la aplicación, tiene un ciclo de vida similar que pasa por varios estados que pueden verse en la figura 7.1 .

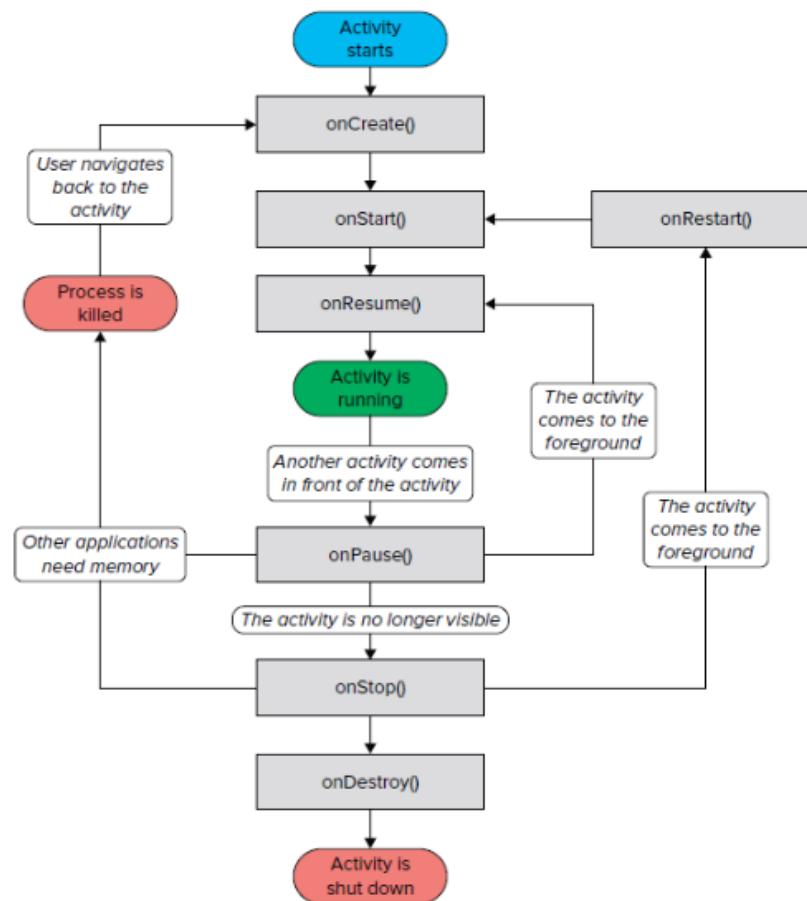


IMAGE REPRODUCED FROM WORK CREATED AND SHARED BY THE ANDROID OPEN SOURCE PROJECT AND USED ACCORDING TO TERMS DESCRIBED IN THE CREATIVE COMMONS 2.5 ATTRIBUTION LICENSE. SEE <http://developer.android.com/reference/android/app/Activity.html>

Figura 7.1: Flujo de vida de una actividad

La actividad se crea en la función `onCreate()`, esta función es similar en todas las actividades de la aplicación, en la figura 7.2 se muestra un ejemplo en código de la función.

```

public class DietHistory extends AppCompatActivity {
    private FirebaseAuth auth;
    private com.dietnow.app.ucm.fdi.adapters.PublishedDietAdapter historyDietAdapter;
    private androidx.recyclerview.widget.RecyclerView RecyclerView;
    private DatabaseReference bd;
    private ArrayList<Diet> dietList;
    private ArrayList<Diet> Dietas;
    private TextView titulo,desc,likes,visit;
    private Button ver;
    private String id, dietId;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_diet_history);
        auth = FirebaseAuth.getInstance();
        RecyclerView = findViewById(R.id.dietHistoryRecycler);
        bd = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
        dietList = new ArrayList<Diet> ();
        Dietas = (ArrayList<Diet>) getIntent().getExtras().getSerializable("Diétas");
        titulo = findViewById(R.id.hDietTitulo);
        desc = findViewById(R.id.hDietDesc);
        likes = findViewById(R.id.hLikesDiet);
        visit = findViewById(R.id.hNVisitDiet);
        ver = findViewById(R.id.hDietShowBtn);
        dietId = "";
        RecyclerView.setLayoutManager(new LinearLayoutManager(context: this));
        //-----Dieta Actual-----
        getDietInfo();
        //-----Fin Dieta Actual-----
        //-----Lista de Dietas-----
        getDiet();
        //-----Fin Lista de Dietas-----
    }
}

```

Figura 7.2: Código de la función onCreate()

La mayoría de las actividades creadas para el desarrollo de la aplicación siguen una serie de pasos:

- Se establece la vista a la que hace referencia la actividad, en este caso es la se hace referencia a la vista de la figura 7.3.
- Se inicializan los elementos de la vista, así como la referencia a la base de datos y al módulo de autenticación.
- En algunas actividades se utilizan funciones que cargan el estado actual de la base de datos para actualizar los valores en la vista, en esta actividad las funciones de getDiet() y getDietInfo() se encargan de cargar los valores que hay en la base de datos. Las figuras 7.4 y 7.5 muestran el código de getDiet() y getDietInfo() respectivamente.



Figura 7.3: Elementos de la vista Dieta Actual

```
private void getDiet() {
    // ver que dieta es la actual para no meterla en la lista de dietas anteriores
    /* */
    bd.child("users").child(auth.getUid()).child("diet").get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
        @Override
        public void onComplete(@NonNull Task<DataSnapshot> task) {
            dietId = task.getResult().getValue(String.class);
            bd.child("diet_history").child(auth.getUid()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
                @Override
                public void onComplete(@NonNull Task<DataSnapshot> task) {
                    for (DataSnapshot ds : task.getResult().getChildren()) {
                        for (Diet d : Dietas) {
                            if (d.getId().equals(ds.getKey()) && !dietId.equals(d.getId())) {
                                dietList.add(d);
                            }
                        }
                    }
                }
            });
            historyDietAdapter = new PublishedDietAdapter(dietList, dietId, context: DietHistory.this, isAdmin: false);
            RecyclerView.setAdapter(historyDietAdapter);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.d( tag: "OnFailureDietHistory: ", msg: "");
            e.printStackTrace();
        }
    });
}

}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Log.d( tag: "OnFailureDietHistory: ", msg: "");
        e.printStackTrace();
    }
});
});
```

Figura 7.4: Código de la función getDiet()

```
private void getDietInfo(){
    bd.child("users").child(auth.getUid()).child("diet").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {

            bd.child("diets").child(snapshot.getValue().toString()).addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    Diet diet = snapshot.getValue(Diet.class);
                    titulo.setText(diet.getTitle());
                    desc.setText(diet.getDescription());
                    id = diet.getId();
                    HashMap<String, Boolean> visita = diet.getVisits();
                    HashMap<String, Boolean> rating = diet.getRating();
                    visit.setText(String.valueOf(visita.size()));
                    Integer counter =0;
                    if(rating != null) {
                        for (Boolean type : rating.values()) {
                            counter += type ? 1 : 0;
                        }
                    }
                    likes.setText(String.valueOf(counter));
                    ver.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            Intent intent = new Intent( packageContext: DietHistory.this, ViewDietActivity.class);
                            intent.putExtra( name: "did", id);
                            startActivity(intent);
                        }
                    });
                }
            });

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
            }
        });
    });
    @Override
    public void onCancelled(@NonNull DatabaseError error) {
```

Figura 7.5: Código de la función getDietInfo()

7.2.2. Iniciar sesión

La primera pantalla de la aplicación corresponde al módulo de login (ver figura 7.6), una vez cargada la pantalla contiene el logotipo de la aplicación, los campos correo electrónico y contraseña para poder iniciar sesión y los botones iniciar sesión y registrarse.

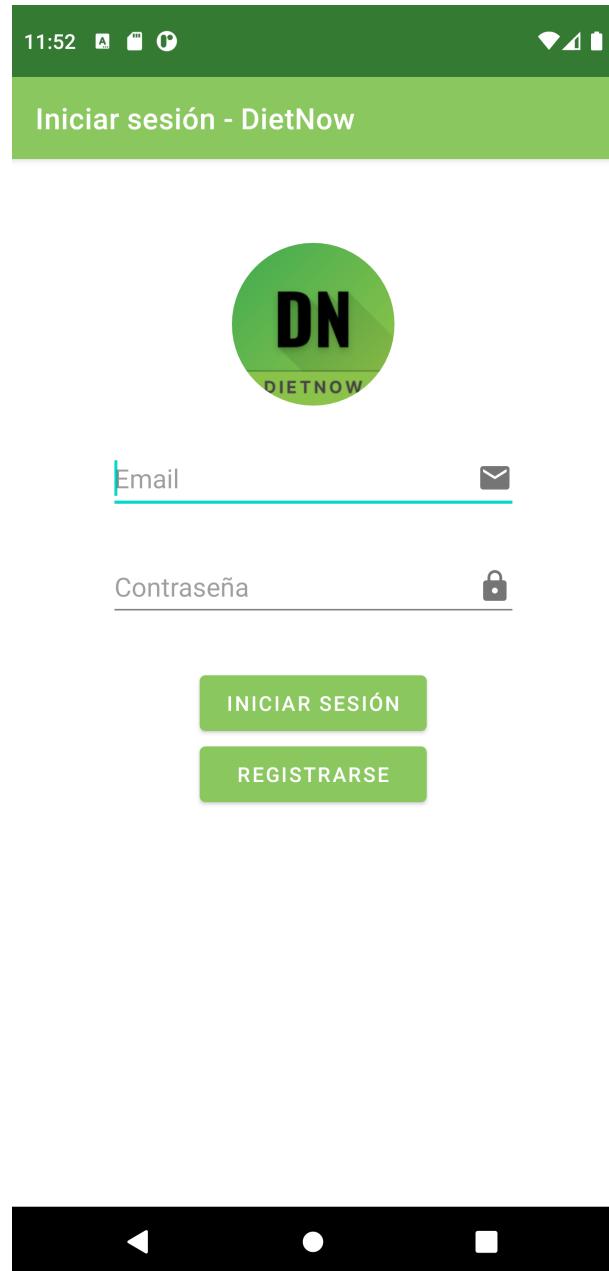


Figura 7.6: Pantalla de login en la aplicación DietNow

En caso de que un usuario quiera darse de alta en la aplicación debe presionar “Registrarse” que le redirigirá a una nueva ventana donde deberá llenar los siguientes campos:

- Email
- Contraseña
- Repetir la contraseña
- Nombre
- Apellidos
- Edad

- Género

La aplicación ofrece soporte para inglés y español (ver figura 7.7). El lenguaje a utilizar se determina durante la carga de la aplicación, la cual detecta el lenguaje utilizado en el dispositivo y lo coteja con la lista de lenguajes disponibles. En caso de no encontrar coincidencias se usa el español como idioma por defecto.

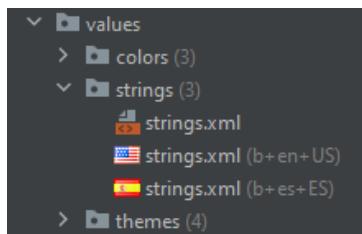


Figura 7.7: Lista de idiomas soportados por la aplicación DietNow

Se pueden incluir otros idiomas de manera sencilla, añadiendo un nuevo fichero con las traducciones en el nuevo idioma. En la figura 7.8 se puede ver un ejemplo de un archivo de traducciones.

```

<string name="settings">Ajustes</string>
<string name="welcome">Bienvenido/a</string>
<string name="login_failed">Credenciales incorrectos</string>
<string name="register_successful">Cuenta creada correctamente</string>
<string name="register_failed">Error al crear cuenta</string>
<string name="password_tooltip">6 caracteres como mínimo</string>
<string name="years">años</string>
<string name="create_account">Crear cuenta</string>
<string name="change">Cambiar</string>
<string name="select_image">Selecciona una imagen</string>
<string name="select_image_error">Error al seleccionar una imagen</string>
<string name="select_doc_error">Error al seleccionar un documento</string>
<string name="my_stats">Mis estadísticas</string>
<string name="uploading">Subiendo</string>
<string name="uploaded_progress">subido</string>
<string name="uploaded_final">Imagen actualizada con éxito</string>
<string name="uploaded_final_doc">Documento subido con éxito</string>
```

Figura 7.8: Fragmento del código del fichero con texto traducido

Registrarse en la aplicación

En caso de que el usuario que se encuentre en el login no tenga cuenta pulsando el botón “Registrarse” accederá a la vista de la figura 7.9 donde debe introducir los campos requeridos para poder registrarse en la aplicación.

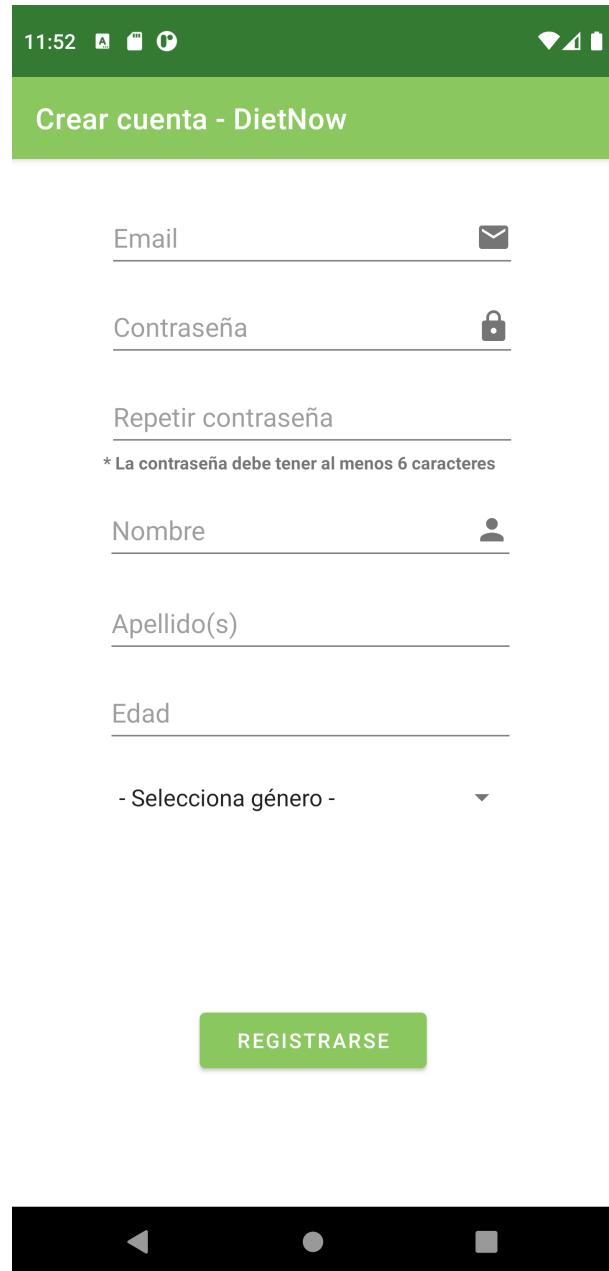


Figura 7.9: Formulario de registro en la aplicación DietNow

Una vez introducidos los campos requeridos y tras pulsar en el botón de registrarse, se inicia sesión con la cuenta creada.

Para realizar la funcionalidad descrita anteriormente se ha creado la actividad de las figuras 7.10 y 7.11:

```
public class RegisterActivity extends AppCompatActivity implements AdapterView.OnItemSelectedListener {

    private Button register;
    private Spinner genres;
    private EditText email;
    private EditText passwd;
    private EditText passwdRepeat;
    private EditText name;
    private EditText lastname;
    private EditText age;
    private FirebaseAuth auth;
    private DatabaseReference mDatabase;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);

        // inicializar Google Firebase
        auth        = FirebaseAuth.getInstance();
        mDatabase   = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();

        // inicializar los componentes por ID
        email      = findViewById(R.id.registerEmail);
        passwd     = findViewById(R.id.registerPassword);
        passwdRepeat = findViewById(R.id.registerPasswordRepeat);
        name       = findViewById(R.id.registerName);
        lastname   = findViewById(R.id.registerLastname);
        age        = findViewById(R.id.registerAge);
        progressBar = findViewById(R.id.progressBarAdmin);

        // change genre action
        genres = findViewById(R.id.registerGenre);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource( context: RegisterActivity.this, R.array.genres, android.R.layout.simple_spinner_item );
        adapter.setDropDownViewResource(android.R.layout.simple_dropdown_item);
        genres.setAdapter(adapter);
        genres.setOnItemSelectedListener(this);

        // register button action
    }
}
```

Figura 7.10: Fragmento de código de la creación de la actividad Register

```

register.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        progressBar.setVisibility(View.VISIBLE);

        Boolean isValid = !email.getText().toString().isEmpty();
        isValid = isValid || !passwd.getText().toString().isEmpty();
        isValid = isValid || !passwdRepeat.getText().toString().isEmpty();
        isValid = isValid || !name.getText().toString().isEmpty();
        isValid = isValid || !age.getText().toString().isEmpty();

        if(isValid){
            if(passwd.getText().toString().equalsIgnoreCase(passwdRepeat.getText().toString())){
                User.UserGender uGender = User.UserGender.NO_GENRE;
                if(!genres.getSelectedItem().toString().isEmpty()){
                    if(genres.getSelectedItem().toString().equalsIgnoreCase("masculino")){
                        uGender = User.UserGender.MALE;
                    } else if(genres.getSelectedItem().toString().equalsIgnoreCase("femenino")){
                        uGender = User.UserGender.FEMALE;
                    } else{
                        uGender = User.UserGender.NO_GENRE;
                    }
                }

                User user = UserService.getInstance().register(email.getText().toString(),
                    name.getText().toString(),
                    lastname.getText().toString(),
                    passwd.getText().toString(),
                    uGender,
                    height: 0.0,
                    Integer.parseInt(age.getText().toString()));

                register(user, passwd.getText().toString());
            } else{
                Toast.makeText(getApplicationContext(),
                    "¡Las contraseñas no coinciden!",
                    Toast.LENGTH_SHORT).show();
            }
        } else{
            Toast.makeText(getApplicationContext(),
                "¡Hay algún campo que está vacío!",
                Toast.LENGTH_LONG).show();
        }
    }
});
}

```

Figura 7.11: Fragmento de código del botón registrar del formulario

En la actividad indicada anteriormente, se hace referencia mediante `setContentView(R.layout.activityRegister)` a la vista indicada en la figura 7.9 y se inicializan las referencias a Realtime Database y a Authentication. En esta última referencia, se guarda la información para el inicio de sesión de los usuarios de la aplicación. Dicha referencia guarda el usuario que se encuentra *logueado* en la aplicación después de haber iniciado sesión. Al igual que en el resto de vistas se inicializan los componentes de la vista, en este caso los botones y los campos de edición de texto. Para el selector de género (ver figura 7.12) se ha utilizado un adaptador que permite seleccionar un elemento dentro de una lista.

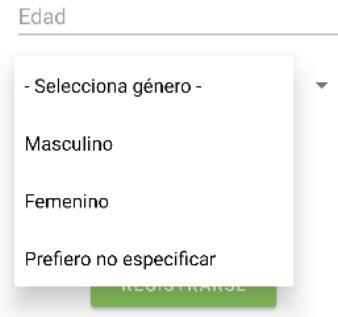


Figura 7.12: Selector de género en el formulario de registro

Posteriormente se establece la acción relativa al botón de registrarse. El usuario debe de haber introducido los campos requeridos correctamente, así como haber introducido la misma contraseña con el tamaño especificado en las dos ocasiones.

Después de haber comprobado que se han introducido los datos con el formato solicitado, se crea un usuario llamando al método `register()` de la clase `UserService` (ver figura 7.13).

```
public class UserService {  
  
    private static UserService instance;  
  
    // return user id  
    public User register(String email, String name, String lastname, String password, User.UserGender gender, Double height, Integer age){  
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
        String created = dateFormat.format(new Date());  
  
        User newUser = new User(email, name, lastname, this.encodePassword(password), gender.name(),  
            height, User.UserRole.USER.name(), age, created, active: true);  
  
        return newUser;  
    }  
}
```

Figura 7.13: Código de la función `register()` de `UserService`

Tras haber creado la instancia del usuario se registrará al usuario en la base de datos (ver figura 7.14).

```

private void register(User user, String rawPassword){
    Query query = mDatabase.child("users").orderByChild("email").equalTo(user.getEmail());
    query.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot){
            if(snapshot.exists()){
                User emailUser = snapshot.getChildren().iterator().next().getValue(User.class);
                progressBar.setVisibility(View.GONE);
                if(emailUser.getActive()){
                    Toast.makeText(getApplicationContext(),
                            "Este email ya ha sido registrado",
                            Toast.LENGTH_SHORT).show();
                }else{
                    Toast.makeText(getApplicationContext(),
                            "Este email ha sido dado de baja contacta con un adminis...",
                            Toast.LENGTH_SHORT).show();
                }
            }else{
                auth.createUserWithEmailAndPassword(user.getEmail(), rawPassword).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if(task.isSuccessful()){
                            FirebaseUser firebaseUser = auth.getCurrentUser();
                            mDatabase.child("users").child(firebaseUser.getUid()).setValue(user);
                            updateUI( success: true);
                        } else{
                            updateUI( success: false);
                        }
                    }
                }).addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Log.d( tag: "OnFailureRegisActivity:", msg: "");
                        e.printStackTrace();
                    }
                });
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {}
    });
}

```

Figura 7.14: Código de la función `register()` del formulario

En dicho método, se accede a la tabla de la base datos de usuario, ordenando los resultados por email para posteriormente comprobar si existe un usuario activo con el email indicado, un administrador ha dado de baja al usuario o no existe ese usuario en la base de datos. Si el usuario no existe en la base de datos se registra al usuario en el módulo de autenticación mediante el método `createUserWithEmailAndPassword()`.

Si se ha registrado correctamente en el módulo de Autenticación, se crea una entrada de ese usuario en Realtime Database en la tabla de usuarios:

`mDatabase.child(users).child(firebaseUser.getUid()).setValue(user);` (ver figura 7.15).

```

auth.createUserWithEmailAndPassword(user.getEmail(), rawPassword).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()){
            FirebaseUser firebaseUser = auth.getCurrentUser();
            mDatabase.child("users").child(firebaseUser.getUid()).setValue(user);
            updateUI( success: true);
        } else{
            updateUI( success: false);
        }
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Log.d( tag: "OnFailureRegisActivity:", msg: "");
        e.printStackTrace();
    }
});
}

```

Figura 7.15: Código la consulta a la base de datos

Después de haber creado al nuevo usuario correctamente se actualiza la vista (ver figura 7.16).

```

private void updateUI(Boolean success){
    if(success){
        Toast.makeText(getApplicationContext(),
            "Cuenta creada correctamente",
            Toast.LENGTH_SHORT).show();

        progressBar.setVisibility(View.INVISIBLE);

        finish();
        Intent intent = new Intent( packageContext: RegisterActivity.this, UserPageActivity.class);
        startActivity(intent);
    } else{
        Toast.makeText(getApplicationContext(),
            "Error al crear cuenta",
            Toast.LENGTH_LONG).show();

        progressBar.setVisibility(View.INVISIBLE);
    }
}

```

Figura 7.16: Código de la función que actualiza la vista

En el método `updateUI()`, se muestra un mensaje de error en el caso de no haber registrado correctamente al usuario y un mensaje de éxito en el caso contrario. Se realiza un `finish()` de la actividad actual, para sacarla de la pila de llamadas y en el caso de pulsar en el botón de ir hacia atrás en el móvil, no mantenga en memoria dicha vista.

Posteriormente se crea un objeto `Intent`, el cual se utiliza para realizar redirecciones entre vistas. En este caso y, después de haber *iniciado sesión* correctamente al usuario, se redirige a la actividad de `UserPageActivity()` (ver figura 7.17) la cual en su método `OnCreate()` hace referencia a la vista de la figura 7.18 que se cargará.

```
public class UserPageActivity extends AppCompatActivity {

    private Button dieta, perfil, dietasCreadas, dietasPub;
    private FirebaseAuth auth;
    private DatabaseReference db;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_page);

        dieta = findViewById(R.id.dietaBtn);
        auth = FirebaseAuth.getInstance();
        db = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
    }
}
```

Figura 7.17: Código de UserPageActivity()



Figura 7.18: Vista del menú principal de un usuario

7.2.3. Crear dieta y modificarla

Esta vista sirve para crear una dieta, añadiendo su título y su descripción. Posteriormente en la vista de modificar dieta se pueden editar los campos anteriormente mencionados así como añadir alimentos de forma manual o con la cámara y subir documentos relativos a la dieta.

Para realizar esta funcionalidad desde la actividad `UserPageActivity`, en el menú principal, mostrado en la figura 7.18, se realiza un `Intent` cuando se pulsa sobre del botón de “Mis dietas creadas“ a la actividad `MyDietsActivity` (ver figura 7.19).

```

dietasCreadas = findViewById(R.id.misDietasBtn);

// Acciones de los componentes
dietasCreadas.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Con Intent podemos "redirigir" al usuario a nueva actividad
        Intent intent = new Intent(getApplicationContext(), MyDietsActivity.class);
        startActivity(intent);
    }
});

```

Figura 7.19: Código del evento asociado al botón “Mis dietas creadas”

Después de haber pulsado el botón desde el método `OnCreate()` de `MyDietsActivity` (ver figura 7.19) se hace referencia a la vista que se muestra en la figura 7.21 para asociar dicha vista con la actividad.

```

public class MyDietsActivity extends AppCompatActivity implements SearchView.OnQueryTextListener{
    private FirebaseAuth auth;
    private SearchView searchDiet;
    private MyDietsAdapter MyDietsAdapter;
    private androidx.recyclerview.widget.RecyclerView RecyclerView;
    private DatabaseReference bd;
    private ArrayList<Diet> dietList;
    private FloatingActionButton newDietBtn;
    private String CurrentUser;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //-----Variables-----
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my_diets);
        auth = FirebaseAuth.getInstance();
        searchDiet = findViewById(R.id.searchMyDiet);
        RecyclerView = findViewById(R.id.MyDietsRecycler);
        bd = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
        dietList = new ArrayList<Diet>();
        newDietBtn = findViewById(R.id.newDietBtn);
        CurrentUser = auth.getCurrentUser().getUid();
        RecyclerView.setLayoutManager(new LinearLayoutManager(context: this));
        //-----Fin Variables-----
        //-----Lista de Dietas-----
        getDiet();
        //-----Fin Lista de Dietas-----
        //-----Search-----
        searchDiet.setOnQueryTextListener(this);
        //-----Fin search-----
        //-----Boton New-----
        newDietBtn.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(), CreateDietActivity.class);
                startActivity(intent);
                finish();
            }
        });
        //-----Fin boton new-----
    }
    private void getDiet(){
        bd.child("diets").addValueEventListener(new ValueEventListener() {
            @Override

```

Figura 7.20: Código de la actividad `MyDietsActivity`

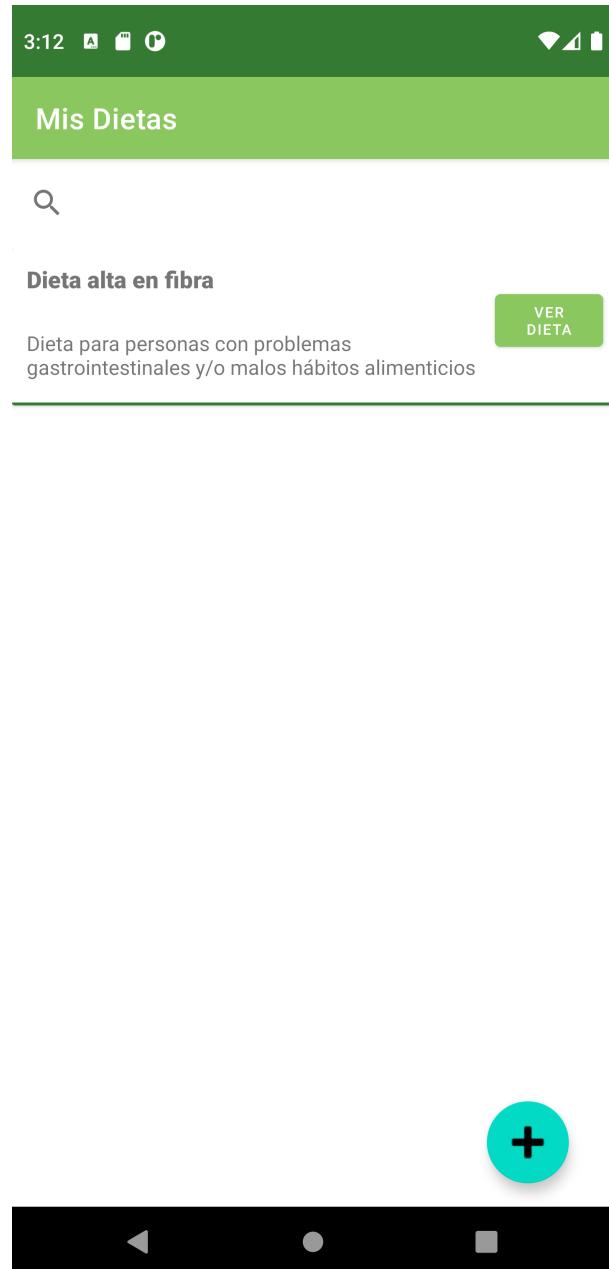


Figura 7.21: Vista “Mis dietas”

En dicha vista se cargan todas las dietas que ha creado el usuario, para ello se crea un **Adapter** para poder ir mostrando dinámicamente las distintas dietas de la aplicación.

En la figura 7.22 se puede ver como se cargan todas las dietas que ha creado el usuario que ha iniciado sesión para posteriormente ir mostrándolas dinámicamente.

```

private void getDiet(){
    bd.child("diets").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            dietList.clear();
            for (DataSnapshot ds : snapshot.getChildren()) {
                String titulo = ds.child("title").getValue().toString();
                Boolean active = ds.child("active").getValue(Boolean.class);
                String descripcion = ds.child("description").getValue().toString();
                String user = ds.child("user").getValue().toString();
                Diet us = new Diet(descripcion, titulo);
                us.setId(ds.child("id").getValue().toString());
                if(active && user.equals(CurrentUser)) {
                    dietList.add(us);
                }
            }
            MyDietsAdapter = new MyDietsAdapter(dietList, context: MyDietsActivity.this);
            RecyclerView.setAdapter(MyDietsAdapter);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.e( tag: "OnFailureMyDietsAc: ", error.toString());
        }
    });
}

```

Figura 7.22: Código de la función que carga las dietas creadas

Además dentro de la vista “Mis Dietas“ (ver figura 7.21) pulsando sobre el botón de “+“ se añade una nueva dieta, lo cual hace que se asocie un Intent a la nueva actividad sobre ese botón. Posteriormente se hace un finish() [26] para sacar de la pila de actividades la actividad actual y cuando se pulse el botón de ir atrás en el móvil no vuelva a esa vista (ver figura 7.23).

```

newDietBtn.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MyDietsActivity.this, CreateDietActivity.class);
        startActivity(intent);
        finish();
    }
});

```

Figura 7.23: Código de la función asociada al botón + de Mis Dietas

En dicha vista (ver figura 7.24) se introduce el nombre y la descripción de la dieta y se pulsa sobre el botón de guardar para actualizar la base de datos con la nueva información (ver figura 7.25).

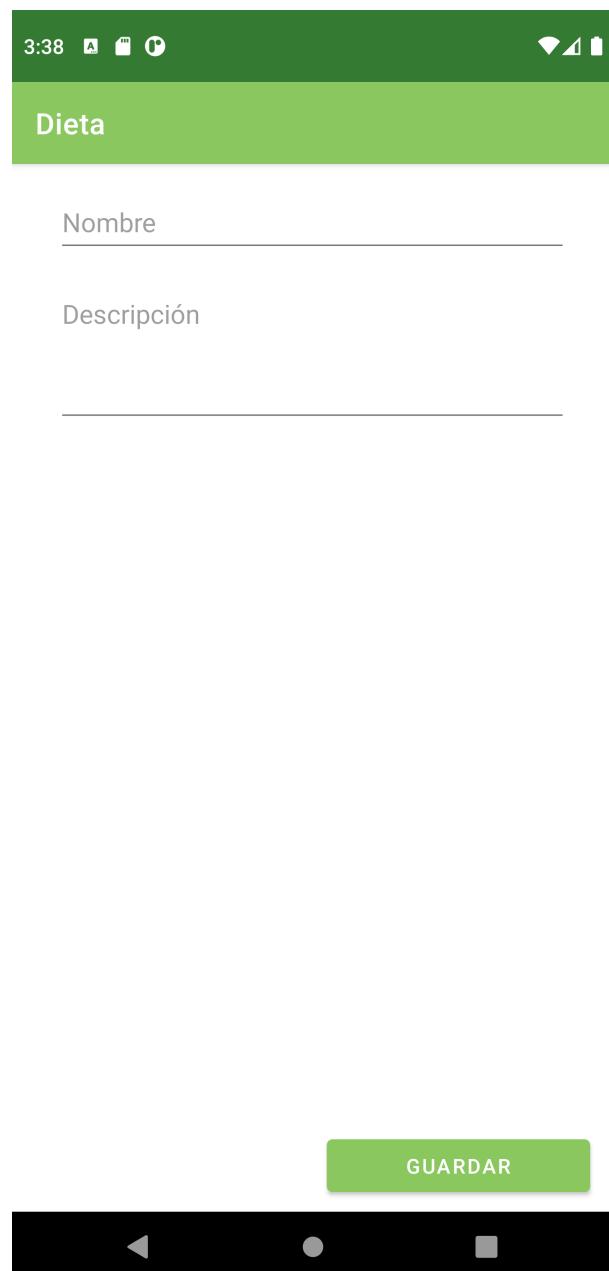


Figura 7.24: Vista de crear nueva dieta

```
// Acciones de los componentes
create.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        progress.setVisibility(View.VISIBLE);

        Diet toCreate = DietService.getInstance().parseDiet(
            title.getText().toString(),
            description.getText().toString(),
            new HashMap<String, Boolean>(), new HashMap<String, Boolean>(),
            kcal: 0.0, active: true, published: false
        );
        uploadDietToFirebase(toCreate, actualDiet);
    }
});
```

Figura 7.25: Código de la función asociada al botón “Guardar“ de crear nueva dieta

Una vez creada la dieta en la vista de mis dietas creadas, si se pulsa sobre ver dieta se muestra la vista de “Ver Dieta“ (ver figura 7.26). En esta vista se puede editar la dieta creada por el usuario así como publicar una dieta para que sea visible para el resto de usuarios de la aplicación. Si la dieta está despublicada solo será vista por el usuario que ha creado la dieta y ningún deportista podrá seguir la dieta.



Figura 7.26: Vista de Ver Dieta

Para poder introducir alimentos o subir documentos a la dieta se pulsa sobre el botón de editar donde se redirige a otra vista (ver figura 7.27).

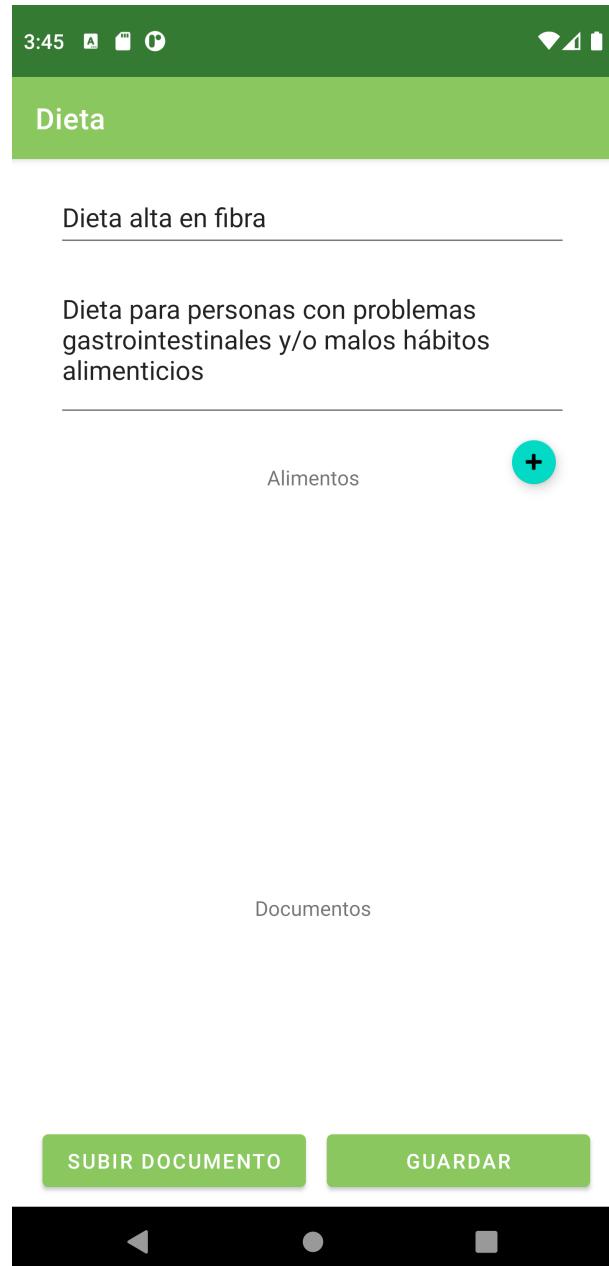


Figura 7.27: Vista de Editar Dieta

Si se pulsa sobre el botón “+“ se pueden añadir alimentos a la dieta. Se pueden añadir alimentos de dos formas diferentes, mediante el uso de la cámara o manualmente. (ver figura 7.28).

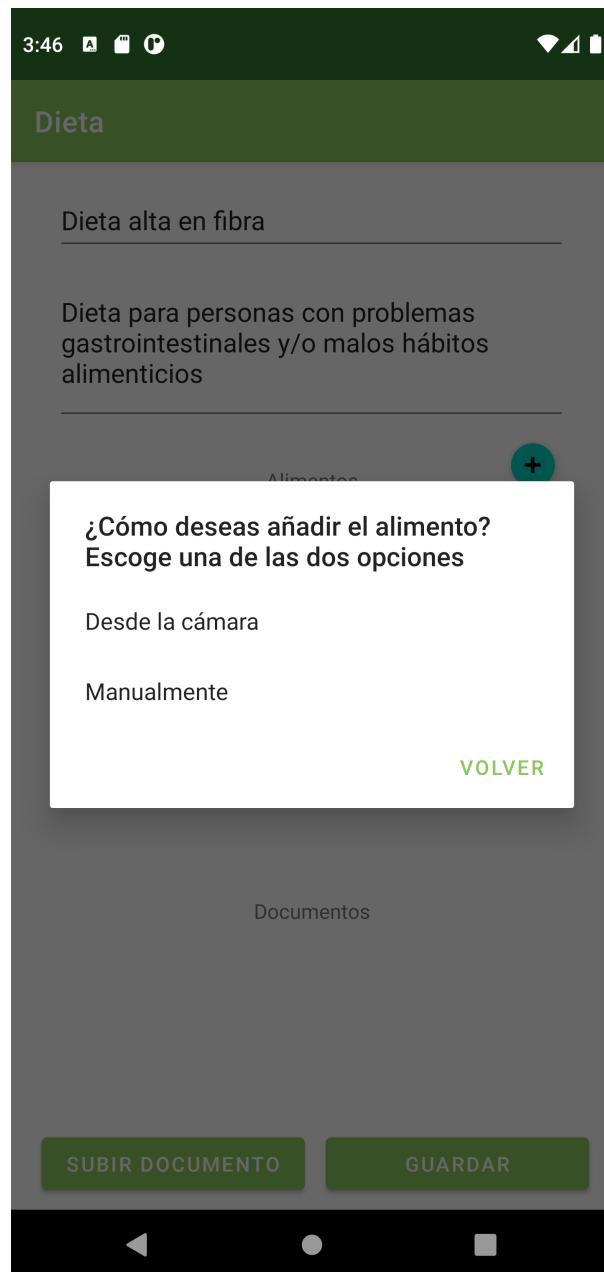


Figura 7.28: Popup que aparece para elegir como añadir un alimento

Si se elige la opción manual (ver figura 7.29), se podrá añadir un alimento de dos maneras:

- Con el código de barras.
- Introduciendo las características principales del alimento.



Figura 7.29: Vista de añadir un alimento manualmente

Para añadir un alimento con el código de barras cuando se pulsa sobre el botón de añadir de la parte superior de la figura 7.29, se utiliza el método de la figura 7.30

```
// Acciones de los componentes
add.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String bc = barcode.getText().toString();
        if (!bc.isEmpty()) {
            GetProductInfo.getInstance().getInfo(bc, actualDiet);
            finish();
        }
    }
});
```

Figura 7.30: Código asociado a añadir alimentos mediante código de barras

La clase `GetProductInfo` (ver figura 7.31) es un Singleton, el cual realiza una petición al alias especial de la interfaz host del bucle invertido [27].

```
public class GetProductInfo {

    private static GetProductInfo instance;
    private DatabaseReference db;
    private Retrofit retrofit;

    public GetProductInfo(){
        db      = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
        retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.2.2:3000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    }

    public static GetProductInfo getInstance(){
        if(instance == null){
            instance = new GetProductInfo();
        }
        return instance;
    }
}
```

Figura 7.31: Código de `GetProductInfo()`

Para obtener la información del código de barras solicitado, se utiliza la API de Open Food Facts (ver figura 7.32).

```

public void getInfo(String barcode, String dietId){
    if(!barcode.isEmpty()){
        OpenFoodFactsService api = retrofit.create(OpenFoodFactsService.class);
        Call<ProductResponse> request = api.getProductInfo(barcode);
        try{
            request.enqueue(new Callback<ProductResponse>() { // la ejecuta async (para sync: execute())
                @Override
                public void onResponse(Call<ProductResponse> call, Response<ProductResponse> response) {
                    if(response.isSuccessful()){
                        ProductResponse apiResponse = response.body();
                        Log.d( tag: "SUCCES", apiResponse.toString());
                        Aliment aliment = new Aliment(apiResponse.getName(), apiResponse.getGrams(), apiResponse.getKcal());

                        // Guardar codigo de barras en el array de la dieta
                        db.child("diets").child(dietId).child("aliments").child(barcode).setValue(aliment);
                        db.child("aliments").child(barcode).setValue(aliment);
                    } else {
                        Log.d( tag: "FAILED", response.message());
                    }
                }

                @Override
                public void onFailure(Call<ProductResponse> call, Throwable t) {
                    Log.d( tag: "FAILED", msg: "FAILED HTTP REQUEST");
                    t.printStackTrace();
                }
            });
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

Figura 7.32: Código de getInfo() dentro de GetProductInfo()

Para añadir el alimento con el código de barras se ha utilizado Retrofit para poder manejar las peticiones HTTP sobre las diferentes API utilizadas en la aplicación.

Se definen las diferentes rutas sobre las que se va a hacer la petición HTTP al servidor de Node.js sobre la API OpenFoodFacts(ver figura 7.33).

```

public interface OpenFoodFactsService {
    @GET("dietnow/api/product/")
    Call<ProductResponse> getProductInfo(@Query("barcode") String barcode);

    @GET("dietnow/api/nutritionalInfo/")
    Call<ProductResponse> getAllProductInfo(@Query("barcode") String barcode);
}

```

Figura 7.33: Código de la interfaz de la API OpenFoodFacts

Para ello se definen los *endpoints* a los cuales Retrofit va a realizar las peticiones, en este caso para obtener las características principales de un producto y para obtener la información nutricional más específica de ese producto (ver figura 7.34).

```

"use strict"

// GENERAL
const dietnow = require("./DietNow");
const config = require("./config");
const http = require("http");
const url = require("url");
const qs = require("querystring");

// CREACION DEL SERVIDOR
http.createServer(function(request, response){
    try {
        const { pathname, query } = url.parse(request.url);

        // Sólo aceptamos de momentos peticiones GET
        if(request.method !== 'GET'){
            response.end(`{"error": "${http.STATUS_CODES[405]}"}`);
        }
        else{
            const { barcode } = qs.parse(query);
            switch(pathname){
                case '/dietnow/api/product/':
                    dietnow.getProductByBarcode(barcode, function(product){
                        response.end(JSON.stringify(product));
                    });
                    break;
                case '/dietnow/api/nutritionalInfo/':
                    dietnow.getAllProductByBarcode(barcode, function(product){
                        response.end(JSON.stringify(product));
                    });
                    break;
                default:
                    response.end(`{"error": "${http.STATUS_CODES[404]}"}`);
            }
        }
    } catch (error) {
        console.error(error);
    }
}).listen(config.port);

```

Figura 7.34: Creación del servidor de node.js

Posteriormente se hacen peticiones a la API de OpenFoodFacts para obtener la información del producto (ver figura 7.35).

```
const https = require('https');

module.exports = {
    getProduct(barcode, callback){
        https.get(`https://world.openfoodfacts.org/api/v0/product/${barcode}.json`, (resp) =>
            let data = '';
            resp.on('data', (chunk) => {
                data += chunk;
            });

            resp.on('end', () => {
                if(callback && typeof callback === 'function'){
                    callback(data);
                }
            });
        }).on("error", (err) => {
            console.error("Error: " + err.message);
        });
    }
};
```

Figura 7.35: Código de la estructura de las peticiones a la API

La API de Open Food Facts devuelve un JSON con la respuesta a la petición realizada y en el código de la figura 7.36 se formatea la información para obtener solo los campos que posteriormente se mostraran al usuario.

```

'use strict';

const openfoodfacts = require('../OpenFoodFacts');

module.exports = {
    getProductByBarcode : function(barcode = '', callback){
        if(barcode){
            openfoodfacts.getProduct(barcode, function(response){
                if(callback && typeof callback === 'function'){
                    response = JSON.parse(response);
                    // DATOS POR CADA 100g
                    let product = {
                        'name' : (response.product && response.product.generic_name) || 'Product',
                        'active' : true,
                        'kcal' : (response.product && response.product.nutriments['energy-kcal']) || 0,
                        'grams' : 100.0
                    };
                    callback(product);
                }
            });
        }
        return { "error" : "Empty barcode" };
    },
    getAllProductByBarcode : function(barcode = '', callback){
        if(barcode){
            openfoodfacts.getProduct(barcode, function(response){
                if(callback && typeof callback === 'function'){
                    response = JSON.parse(response);
                    // DATOS POR CADA 100g
                    let nutritionalInfo = {
                        'name' : response.product.generic_name || 'Alimento',
                        'kcal' : response.product.nutriments['energy-kcal'] || 0,
                        'grams' : 100.0,
                        'fat' : response.product.nutriments['fat_100g'] || 0,
                        'saturatedFat' : response.product.nutriments['saturated-fat_100g'] || 0,
                        'carbs' : response.product.nutriments['carbohydrates_100g'] || 0,
                        'sugar' : response.product.nutriments['sugars_100g'] || 0,
                        'proteins' : response.product.nutriments['proteins_100g'] || 0,
                        'salt' : response.product.nutriments['salt_100g'] || 0
                    };
                    callback(nutritionalInfo);
                }
            });
        }
        return { "error" : "Empty barcode" };
    }
}

```

Figura 7.36: Código del formateo de los JSON devueltos por OpenFoodFacts

Para crear un alimento de forma manual introduciendo las principales características se utiliza el método mostrado en la figura 7.37

```

addManual.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(!name.getText().toString().isEmpty() && !kcal.getText().toString().isEmpty() && !grams.getText().toString().isEmpty()){
            String pname = "";
            try {
                name = name.getText().toString();
                MessageDigest md5Digest = MessageDigest.getInstance("MD5");
                md5Digest.update(pname.getBytes());
                byte[] digest = md5Digest.digest();
                StringBuffer hexString = new StringBuffer();
                for (int i = 0; i < digest.length; i++) {
                    hexString.append(Integer.toHexString(i & 0xFF & digest[i]));
                }
                pname = hexString.toString();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        Aliment aliment = new Aliment(name.getText().toString(), Double.parseDouble(grams.getText().toString()), Double.parseDouble(kcal.getText().toString()));

        db.child("diets").child(actualDiet).child("aliments").child(pname).setValue(aliment).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                diet_added = true;
                update_and_refresh();
            }
        });

        db.child("aliments").child(pname).setValue(aliment).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                aliment_added = true;
                update_and_refresh();
            }
        });
    }
})
}

```

Figura 7.37: Código del método que añade alimento de forma manual

Se genera un código MD5 para poder insertar el alimento a la base de datos con el código como clave principal.

Para añadir un alimento utilizando la cámara (ver figura 7.38), se realiza mediante la actividad `CameraActivity` que implementa un escáner [28] que al detectar un código de barras identifica el número y realiza la consulta a OpenFoodFacts con el código. En la figura 7.39 se ilustra el código de la `CameraActivity`.



Figura 7.38: Añadir alimento mediante el uso de la cámara

```
public class CameraActivity extends AppCompatActivity {
    private CodeScanner mCodeScanner;
    private int CAMERA_REQUEST_CODE = 101;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_camera);
        setUpPermissions();
        CodeScannerView scannerView = findViewById(R.id.scanner_view);
        mCodeScanner = new CodeScanner(context: this, scannerView);
        mCodeScanner.setDecodeCallback(new DecodeCallback() {
            @Override
            public void onDecoded(@NotNull final Result result) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(context: CameraActivity.this, result.getText(), Toast.LENGTH_SHORT).show();
                    }
                });
            }
        });
        scannerView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { mCodeScanner.startPreview(); }
        });
    }

    @Override
    protected void onResume() {
        super.onResume();
        mCodeScanner.startPreview();
    }

    @Override
    protected void onPause() {
        mCodeScanner.releaseResources();
        super.onPause();
    }

    private void setUpPermissions(){
        int permission = ContextCompat.checkSelfPermission(context: this, Manifest.permission.CAMERA);
        if(permission != PackageManager.PERMISSION_GRANTED){
            makeRequest();
        } else{
        }
    }

    private void makeRequest(){
        ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.CAMERA}, CAMERA_REQUEST_CODE);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NotNull String[] permissions, @NotNull int[] grantResults) {
        if(requestCode == CAMERA_REQUEST_CODE) {
            if(grantResults == null || grantResults[0] != PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(context: this, text: "Se necesitan los permisos de la cámara para poder leer un código de barras", Toast.LENGTH_SHORT);
            } else {
                super.onRequestPermissionsResult(requestCode, permissions, grantResults);
                /* Success */
            }
        }
    }
}
```

Figura 7.39: Código de CameraActivity

Para añadir un documento se utiliza el método de la figura 7.40 que abre los archivos del teléfono para poder seleccionar un documento que este almacenado en el dispositivo (ver figura 7.41).

```
private void getDocFromAndroid(){
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("application/pdf");
    intent.putExtra( name: "crop", value: "true");
    intent.putExtra( name: "return-data", value: true);

    Intent chooserIntent = Intent.createChooser(intent, "Selecciona una imagen");

    startActivityForResult(chooserIntent, PICK_DOC);
}
```

Figura 7.40: Código del método abre los archivos del teléfono para obtener los documentos

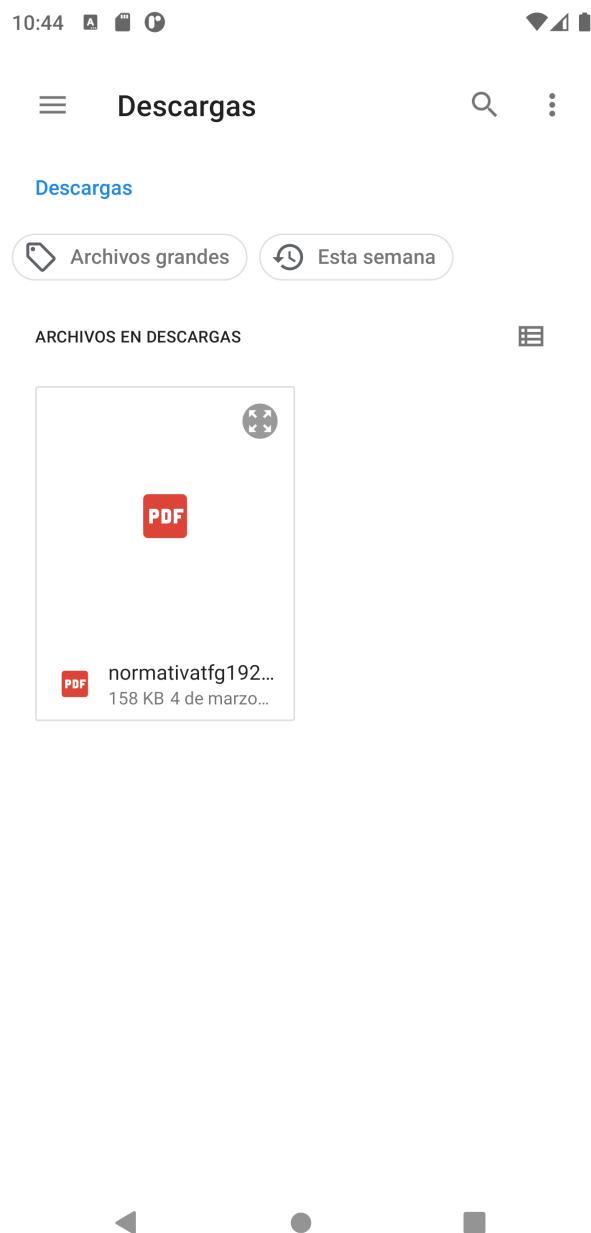


Figura 7.41: Vista del almacenamiento del dispositivo

7.2.4. Dieta actual

Esta vista ofrece al usuario toda la información relativa a la dieta que está siguiendo, en dicha vista se puede visualizar la cantidad consumida de cada alimento a lo largo de la semana, pudiendo introducir la cantidad consumida en el día actual. Además permite valorar la dieta por medio de un me gusta o no me gusta, así como realizar un comentario sobre la dieta, el cual será visto por todos los usuarios.

La funcionalidad de este caso de uso se realiza en la actividad DietInfoActivity (ver figura 7.42).

```
public class DietInfoActivity extends AppCompatActivity {

    private FirebaseAuth auth;
    private DatabaseReference db;
    private StorageReference storageRef;
    private Button monday, tuesday, wednesday, thursday, friday, saturday, sunday, comment, AbandonDiet, like, dislike, guardar;
    private CheckBox checkBox;
    private TextView aliment_id, kcal_info, diet_description, diet_title;
    private EditText info_cantidad;
    private String dietId, dayName; // ID de la dieta que está siguiendo
    private HashMap<String, Aliment> original_aliments;

    private ArrayList<Aliment> alimentList;
    private ArrayList<Button> listaBotones;

    private com.dietnow.app.ucm.fdi.adapters.DietFollowedAdapter dietFollowedAdapter;
    private androidx.recyclerview.widget.RecyclerView RecyclerView;

    @RequiresApi(api = Build.VERSION_CODES.O)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_diet_info);
        getSupportActionBar().setTitle("Información de la dieta");

        // Atributos Firebase
        auth      = FirebaseAuth.getInstance();
        db        = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
        storageRef = FirebaseStorage.getInstance().getReference(); // crear una instancia a la referencia del almacenamiento

        // Atributos de la vista
        alimentList = new ArrayList<Aliment> ();
        listaBotones = new ArrayList<Button>();

        like      = findViewById(R.id.actualDietLike);
        dislike   = findViewById(R.id.actualDietDislike);
        monday    = findViewById(R.id.monday_button);
        tuesday   = findViewById(R.id.tuesday_button);
        wednesday = findViewById(R.id.wednesday_button);
        thursday  = findViewById(R.id.thursday_button);
        friday    = findViewById(R.id.friday_button);
        saturday  = findViewById(R.id.saturdau_button);
        guardar   = findViewById(R.id.GuardarAlimentoDietaBtn);
        sunday    = findViewById(R.id.sunday_button);
        comment   = findViewById(R.id.comment_diet);
        checkBox  = findViewById(R.id.id_checkBox);
        aliment_id = findViewById(R.id.id_aliment);
        kcal_info = findViewById(R.id.id_kcal);
        info_cantidad= findViewById(R.id.id_cantidad);
        AbandonDiet = findViewById(R.id.desuscribirBtn);
        diet_title = findViewById(R.id.diet_name);
        diet_description = findViewById(R.id.viewDietDescription);
        RecyclerView = findViewById(R.id.diet_followed_aliment);
        RecyclerView.setLayoutManager(new LinearLayoutManager( context: this));
    }
}
```

Figura 7.42: Código de DietInfoActivity

En el método `onCreate()` se inicializan las referencias de Realtime Database, Authentication de Firebase y de los diferentes componentes de la vista “Información de la dieta” (ver figura 7.43).



Figura 7.43: Vista Información de la dieta

Para obtener la información de las calorías consumidas durante el día actual se utiliza el método `getAliments()` (ver figura 7.44).

```

private void getAliments() {
    db.child("users").child(auth.getUid()).child("diet").get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
        @Override
        public void onComplete(@NonNull Task<DataSnapshot> task) {
            dietId = task.getResult().getValue(String.class);
            db.child("diets").child(task.getResult().getValue().toString()).child("aliments").get().addOnCompleteListener<DataSnapshot>() {
                @Override
                public void onComplete(@NonNull Task<DataSnapshot> task) {
                    alimentList.clear();
                    for (DataSnapshot ds : task.getResult().getChildren()) {
                        Aliment alimento = ds.getValue(Aliment.class);
                        alimento.setId(ds.getKey());
                        alimentList.add(alimento);
                        originalAliments.put(ds.getKey(), alimento);
                    }

                    dietFollowedAdapter = new DietFollowedAdapter(alimentList, dietId, can_edit, context: DietInfoActivity.this);
                    RecyclerView.setAdapter(dietFollowedAdapter);
                }
            }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.d(tag: "OnFailureDietInfo: ", msg: "");
                    e.printStackTrace();
                }
            });
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.d(tag: "OnFailureDietHistory: ", msg: "");
            e.printStackTrace();
        }
    });
}

```

Figura 7.44: Código del método getAliments()

En dicho método se carga la información relativa a la dieta actual y se rellena la lista de alimentos para que el DietFollowAdapter muestre dinámicamente los alimentos de la dieta (ver figura 7.45).

```

public class DietFollowedAdapter extends RecyclerView.Adapter<DietFollowedAdapter.ViewHolder> {
    private ArrayList<Aliment> localDataSet;
    private ArrayList<Aliment> allAliments;
    private Context context;
    private DatabaseReference db;
    private FirebaseAuth auth;
    private String dietID;
    private ArrayList<Pair<String, ViewHolder>> alimentList_toInsert;
    private Boolean can_edit = false;

    public DietFollowedAdapter(ArrayList<Aliment> dataSet, String diet_id, Boolean can_edit, Context context) {
        localDataSet = dataSet;
        alimentList_toInsert = new ArrayList<>();
        allAliments = new ArrayList<>();
        allAliments.addAll(localDataSet);
        this.context = context;
        db = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
        auth = FirebaseAuth.getInstance();
        dietID = diet_id;
        this.can_edit = can_edit; // para saber si solo es para visualizar los alimentos de otro dia o es para editar los alimentos del dia actual
        // TODO si queremos que no se vea el checkbox habría que ocultarlo y habría que ocultar el boton de guardar de la vista general
    }

    @Override
    public DietFollowedAdapter.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        // Create a new view, which defines the UI of the list item
        View view = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.diet_followed_aliment_item, viewGroup, attachToRoot: false);
        return new DietFollowedAdapter.ViewHolder(view);
    }
}

```

Figura 7.45: Código del adapter DietFollowAdapter()

En el método `OnBindViewHolder()` del adaptador (ver figura 7.46), si se puede editar la cantidad de gramos consumidos por el usuario de un alimento de la dieta, es decir, si se ha seleccionado en la vista el día actual, se consulta la tabla “`diet_history`” de Realtime Database para comprobar si se ha consumido alguna cantidad del alimento seleccionado previamente durante el día actual, para ir guardando ese alimento en un mapa e ir actualizando el número de gramos consumidos de ese alimento. En el caso de que no se pueda editar, es decir no se ha seleccionado el día actual, se actualiza la vista con la cantidad consumida de cada alimento el día seleccionado.

```

public void onBindViewHolder(@NonNull ViewHolder holder, @SuppressLint("RecyclerView") int position) {
    if(can_edit){
        holder.checkBox.setVisibility(View.GONE);
        holder.info_cantidad.setVisibility(View.GONE);
    }
    holder.aliment_id.setText(localDataSet.get(position).getName());
    holder.kcal_info.setText(localDataSet.get(position).getKcal() + " kcal/100g");
    holder.info_cantidad.setText("0");
    holder.aliment_barcode.setText(localDataSet.get(position).getId());
    if(can_edit){
        db.child("diets").child(dietID).child("aliments").child(localDataSet.get(position).getId()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
            @Override
            public void onComplete(@NonNull Task<DataSnapshot> task) {
                holder.totalGR.setText(task.getResult().child("grams").getValue(Long.class).toString() + " gr");
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.d(tag, "OnFailureAdapter: ", msg);
                e.printStackTrace();
            }
        });
    }
    db.child("diet_history").child(auth.getUid()).child(dietID).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
        @RequiresApi(api = Build.VERSION_CODES.O)
        @Override
        public void onComplete(@NonNull Task<DataSnapshot> task) {
            LocalTime currentDate = LocalDateTime.now();
            String strDate = currentDate.format(DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss"));
            HashMap<String, Integer> map_grams_counter = new HashMap<String, Integer>();
            for(DataSnapshot ds : task.getResult().getChildren()) {
                String strDate_info[] = strDate.split(" ");
                String db_date[] = ds.getKey().split(" ");
                if(strDate_info[0].equalsIgnoreCase(db_date[0])) {
                    for(DataSnapshot ds2 : ds.getChildren()) {
                        Integer counter = map_grams_counter.get(ds2.getKey());
                        if(counter != null) { //ya esta ese id en el mapa
                            Integer total = counter + Integer.parseInt(ds.child(ds2.getKey()).getValue().toString());
                            map_grams_counter.put(ds2.getKey(), total);
                        } else {
                            map_grams_counter.put(ds2.getKey(), Integer.parseInt(ds2.getValue().toString()));
                        }
                    }
                }
            }

            if(map_grams_counter.get(holder.aliment_barcode.getText().toString()) == null){
                holder.llevasGR.setText("0");
            }else{
                holder.llevasGR.setText(map_grams_counter.get(holder.aliment_barcode.getText().toString()));
            }
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.d(tag, "OnFailureCreateDiet: ", msg);
            e.printStackTrace();
        }
    });
} else{
    Double d = localDataSet.get(position).getGrams_consumed();
    holder.llevasGR.setText(d.toString());
}

db.child("diets").child(dietID).child("aliments").child(localDataSet.get(position).getId()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
    @Override
    public void onComplete(@NonNull Task<DataSnapshot> task) {
        holder.totalGR.setText(task.getResult().child("grams").getValue(Long.class).toString() + " gr");
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Log.d(tag, "OnFailureAdapter: ", msg);
        e.printStackTrace();
    }
});
}

```

Figura 7.46: Código de OnBindViewHolder()

Cuando el usuario quiere actualizar la cantidad consumida de un alimento de la dieta durante el día actual, debe llenar cuadro de texto (elemento número 1 en la figura 7.48) y posteriormente pulsar sobre el *checkBox* (elemento número 2 en la figura 7.48) cuyo código está ilustrado en la figura 7.47, para indicar la cantidad de gramos de cada alimento a insertar. Desde el `DietFollowedAdapter()` se inserta en una lista de pares el código del alimento a introducir y la información relativa a ese alimento.

```
holder.checkBox.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(holder.checkBox.isChecked())
            alimentList_toInsert.add(new Pair<String,ViewHolder>(holder.aliment_barcode.getText().toString(),holder));
        else{
            alimentList_toInsert.remove(new Pair<String,ViewHolder>(holder.aliment_barcode.getText().toString(),holder));
        }
    }
});
```

Figura 7.47: Código del *listener* del *checkBox* de cada alimento

Después de haber introducido los gramos consumidos de cada alimento y haber pulsado sobre el *checkBox* en cada uno de los alimentos que se quieren actualizar, desde la actividad `DietInfoActivity()`, tras pulsar sobre el botón de guardar (elemento número 3 en la figura 7.48) de la vista “Información de la dieta“ se actualizan los datos en la base de datos (ver figura 7.49).



Figura 7.48: Vista de información de dieta con los pasos para actualizar gramos consumidos

```

public void guardar(){
    if(!alimentList_toInsert.isEmpty()) {
        for (Pair<String, ViewHolder> data : alimentList_toInsert) {
            db.child("users").child(auth.getUid()).child("diets").addValueEventListener(new ValueEventListener() {
                @RequiresApi(api = Build.VERSION_CODES.O)
                @Override
                public void onDataChange(@NotNull DataSnapshot snapshot) {
                    LocalDatetime currentDate = LocalDatetime.now();
                    String strDate = currentDate.format(DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss"));
                    db.child("diet_history").child(auth.getUid()).child(snapshot.getValue().toString()).child(strDate).child(data.first).setValue(Integer.parseInt(data.second.info_cantidad.getText().toString()));
                }
            });
        }
        alimentList_toInsert.clear();
    } else{
        Toast.makeText(context, "Selecciona los alimentos que quieres guardar", Toast.LENGTH_LONG).show();
    }
}

```

Figura 7.49: Código del método `guardar()` que almacena los cambios en la base de datos

En el método de guardar se insertan en la tabla de “`diet_history`“ los alimentos que han sido guardados previamente en la lista de pares, añadiendo una nueva entrada en la base de datos con la fecha actual, con el código del alimento como clave y la cantidad de gramos del alimento consumido como valor.

7.2.5. Añadir comentarios

Esta funcionalidad se creó para que las dietas tuvieran opiniones de los deportistas que se habían suscrito a esa dieta y la habían realizado durante un tiempo. Esta utilidad sirve para conocer cómo otros usuarios han llevado a cabo la dieta, ver si han obtenido los resultados esperados, así como comentar las posibles mejoras de la dieta.

Solo los deportistas que siguen la dieta pueden comentar sobre ella, así como los administradores de la aplicación. Los comentarios se realizan sobre la vista de dieta seguida.

En esta vista (ver figura 7.43) si se pulsa sobre el botón de comentar, aparecen todos los comentarios de esta dieta así como la posibilidad de escribir un nuevo comentario para la dieta (ver figura 7.50).

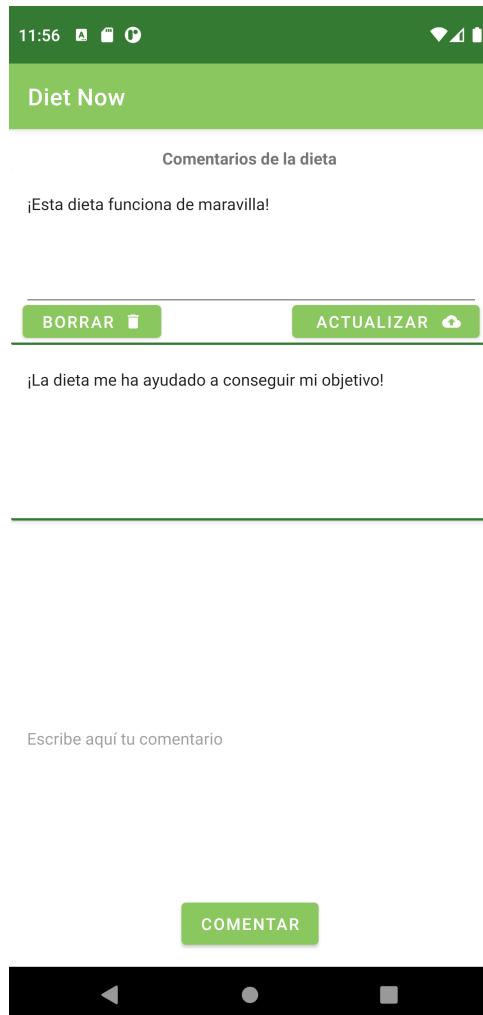


Figura 7.50: Vista del módulo comentarios

Para realizar esta funcionalidad se hace un `Intent`, redireccinando al usuario a la vista

de los comentarios (ver figura 7.51).

```
comment.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(packageContext: DietInfoActivity.this, DietComments.class);
        intent.putExtra(name: "did", dietId);
        startActivity(intent);
    }
});
```

Figura 7.51: Código del método comment() que redirige a la vista de los comentarios

En este Intent además, se le pasa por parámetro a la vista el id de la dieta mediante el método intent.putExtra("did", dietId), para que en la vista de comentarios ya se tenga guardado el id de la dieta sin necesidad de consultar la base de datos. En el método OnCreate() de la actividad DietComments, se guarda en una variable el id de la dieta para ser utilizado posteriormente al insertar comentarios en esa dieta (ver figura 7.52).

```
public class DietComments extends AppCompatActivity {

    private String actualDiet;
    private FirebaseAuth auth;
    private DatabaseReference db;
    private EditText comment;
    private Button commentBtn;
    private CommentsAdapter commentsAdapter;
    private ArrayList<Comment> commentList;
    private androidx.recyclerview.widget.RecyclerView RecyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_diet_comments);

        // Atributos Firebase
        auth      = FirebaseAuth.getInstance();
        db       = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();

        // Atributos de la vista
        actualDiet = getIntent().getExtras().getString(key: "did");
        comment   = findViewById(R.id.newCommentText);
        commentBtn = findViewById(R.id.newCommentBtn);
        RecyclerView = findViewById(R.id.allDietComments);
        commentList = new ArrayList<Comment>();

        RecyclerView.setLayoutManager(new LinearLayoutManager(context: this));
    }
}
```

Figura 7.52: Código de la clase DietComments()

Mediante el método “getIntent().getExtras().getString(“did”)”, guarda en una variable el id de la dieta. En el caso de que se le hubieran pasado más parámetros a la

actividad, solo habría que cambiar el string con el que se hace “`intent.putExtra()`” y el string referenciado en la actividad que recibe los argumentos por medio del Intent.

En la vista de comentarios (ver figura 7.50) se pueden realizar varias acciones:

- a). Se puede escribir un nuevo comentario sobre la dieta.
- b). Se puede actualizar un comentario previamente realizado por el usuario *logueado* en la aplicación.
- c). Se pueden borrar comentarios previamente realizados por el usuario *logueado* en la aplicación.

Para implementar las funcionalidades descritas anteriormente en la actividad `DietComments`, en el método `OnCreate()` de la actividad se llama al método `getComments()` que carga de base de datos los comentarios de la dieta (ver figura 7.53).

```
private void getComments(){
    db.child("comments").child(actualDiet).addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            commentList.clear();
            for(DataSnapshot ds : snapshot.getChildren()){
                Comment comment = ds.getValue(Comment.class);
                comment.setId(ds.getKey());
                commentList.add(comment);
            }

            commentsAdapter = new CommentsAdapter(commentList, context: DietComments.this, actualDiet);
            RecyclerView.setAdapter(commentsAdapter);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.e( tag: "OnCancelDietComments: ",error.toString());
        }
    });
}
```

Figura 7.53: Código del `getComments()`

En la tabla comentarios, se recorren todos los comentarios relativos a la dieta actual para posteriormente ir insertándose en una lista, que posteriormente utilizará el adaptador `CommentsAdapter` para mostrar dinámicamente los distintos comentarios de la dieta.

En el método `OnBindViewHolder()` del adaptador `CommentsAdapter` se cargan los datos relativos a cada uno de los comentarios de la dieta, para poder ser visualizados posteriormente por el usuario (ver figura 7.54).

```

public void onBindViewHolder(@NonNull CommentsAdapter.ViewHolder holder, @SuppressLint("RecyclerView") int position) {
    holder.id.setText(localDataSet.get(position).getId());
    holder.comment.setText(localDataSet.get(position).getComment());

    String user_comment_id = localDataSet.get(position).getUser();
    FirebaseUser user = auth.getCurrentUser();
    db.child("users").child(user.getUid()).addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {

            if(!user.getUid().equalsIgnoreCase(user_comment_id) && !snapshot.child("role").getValue(String.class).equalsIgnoreCase(anotherString: "admin")) {
                holder.edit.setVisibility(View.GONE);
                holder.delete.setVisibility(View.GONE);
                holder.comment.setFocusable(false);
                holder.comment.setFocusableInTouchMode(false);
                holder.comment.setInputType(InputType.TYPE_NULL);
                holder.comment.setBackgroundColor(Color.TRANSPARENT);
                holder.comment.setHeight(150);
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
        }
    });
}

```

Figura 7.54: Código del método `OnBindViewHolder()` del adaptador `CommentsAdapter`

En el caso de que el usuario quiera insertar un comentario en la dieta, se insertará en comentario en la base de datos con el método `uploadComment()` (ver figura 7.55).

```

private void uploadComment(){
    SimpleDateFormat dateFormat = new SimpleDateFormat(pattern: "y-M-d H:m:s");
    String created = dateFormat.format(new Date());
    String commentId = db.child("comments").child(actualDiet).push().getKey();
    Comment c = new Comment(auth.getUid(), comment.getText().toString(), created);
    c.setId(commentId);

    db.child("comments").child(actualDiet).child(commentId).setValue(c).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            Toast.makeText(getApplicationContext(), "Comentario de dieta creado correctamente", Toast.LENGTH_SHORT).show();
            getComments();
            comment.setText("");
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.d(tag: "OnFailureDietComments: ", msg: "");
            e.printStackTrace();
        }
    });
}

```

Figura 7.55: Código del método `uploadComment()`

Las acciones para borrar y actualizar comentarios previamente realizados, se realizan en el `CommentsAdapter` ya que este contiene la información relativa a cada comentario, así como los botones de actualizar y guardar de cada comentario. En el caso de querer actualizar el comentario se realizará tras haber cambiado el texto del comentario después de haber pulsado el botón de actualizar relativo al comentario (ver figura 7.56). En el caso de querer borrar el comentario, la actualización se llevará a cabo después de pulsar sobre el botón de borrar el comentario (ver figura 7.56).

```

holder.edit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        db.child("comments").child(diet_id).child(localDataSet.get(position).getId())
            .child("comment").setValue(holder.comment.getText().toString())
            ;
        Toast.makeText(context, "Comentario actualizado correctamente", Toast.LENGTH_SHORT).show();
    }
});

holder.delete.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        confirmAndDeleteComment(holder);
    }
});

```

Figura 7.56: Código de los métodos asociados a editar y borrar comentario

Cada vez que un comentario es actualizado, insertado o borrado, la vista de los comentarios se actualiza, esto es posible ya que en la actividad `DietComments` se establece un objeto de escucha con el método `addValueEventListener()` sobre la tabla comentarios, para que cuando se modifique una referencia de esa tabla, actualice la información relativa a la vista (ver figura 7.53).

7.2.6. Perfil del usuario

Esta vista ofrece la posibilidad de visualizar información acerca del usuario que ha iniciado sesión en la aplicación así como modificar los datos del perfil. Independientemente del rol que tenga el usuario, en el menú principal existe un botón llamado “Mi Perfil” que redirige al perfil del usuario que ha iniciado sesión (ver figura 7.18).

El módulo perfil de usuario contiene los datos personales del usuario, las gráficas asociadas a su actividad y opciones de administración de cuenta cómo editar o eliminar la cuenta o cerrar sesión.

La ventana de perfil de usuario se puede dividir en las siguientes partes (ver figura 7.57):

- **Información personal:** corresponde a la parte superior de la pantalla, en esta parte se muestran los datos personales del usuario como el nombre, apellido, edad, email, la fecha desde la cual es miembro en la aplicación y la imagen de perfil, además de 2 botones, uno para cambiar la imagen de perfil y otro para ver la gráfica que la dieta actual.
- **Gráficas de pesos y pasos:** mediante el botón deslizante podemos activar las gráficas de pesos y pasos que muestran la evolución de en el tiempo de estos dos registros. El botón circular con el signo “+” permite añadir pasos y/o pesos para el día actual.

- **Acciones:** pulsando sobre el botón de ajustes (imagen de 3 puntos verticales) se despliega un selector con las siguientes opciones:
 - **Historial de dietas:** despliega una nueva ventana donde se muestran por orden cronológico las dietas que ha ido siguiendo el usuario y la dieta que sigue actualmente, al lado de cada dieta hay un botón que para abrir el detalle de la dieta en cuestión.
 - **Cerrar sesión:** cierra la sesión del usuario y vuelve a la pantalla de login.
 - **Editar perfil:** despliega una nueva ventana donde se pueden editar la información personal del usuario.
 - **Eliminar perfil:** despliega una ventana emergente avisando que la eliminación del perfil no se puede deshacer, si el usuario accede se elimina el perfil.



Figura 7.57: Vista de Mi Perfil dividida en 3 secciones

A nivel de código, esta ventana se crea con el método `onCreate()` de “`UserProfileActivity`” (ver figura 7.58) Dentro de `onCreate()` también se declaran todos los listeners de los botones como por ejemplo el de añadir pesos y/o pasos (ver figura 7.59).

```
public class UserProfileActivity extends AppCompatActivity {

    // Necesario para saber cuando el usuario ya ha elegido una imagen de la galeria
    private static final Integer PICK_IMAGE = 1;
    private TextView name, age, email, date;
    private FloatingActionButton addProfile;
    private ImageView image;
    private Uri filePath;
    private Button change, current_diet_graphic;
    private String uid, diet_id;
    private DatabaseReference db;
    private FirebaseAuth auth;
    private FirebaseStorage storage;
    private StorageReference storageRef, imagesRef;
    private AnyChartView steps;
    private AnyChartView weights;
    private CompoundButton selectorStepsWeight;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_profile);

        // inicializar los elementos
        addProfile = findViewById(R.id.addProfile);
        name = findViewById(R.id.profileName);
        age = findViewById(R.id.profileAge);
        email = findViewById(R.id.profileEmail);
        date = findViewById(R.id.profileDate);
        image = (ImageView) findViewById(R.id.profileImage);
        change = findViewById(R.id.profileChangeImg);
        selectorStepsWeight = findViewById(R.id.selectorStepsWeight);
        current_diet_graphic = findViewById(R.id.current_diet_history);
        diet_id = "";

        // inicializar Google Firebase
        auth = FirebaseAuth.getInstance();
        db = FirebaseDatabase.getInstance(MainActivity.FIREBASE_DB_URL).getReference();
        storageRef = FirebaseStorage.getInstance().getReference();
        imagesRef = storageRef.child("images");

        FirebaseUser currentUser = auth.getCurrentUser();
        this.uid = currentUser.getUid();
    }
}
```

Figura 7.58: Código de “UserProfileActivity”



Figura 7.59: Selector de pasos o pesos

Dependiendo de lo que haya elegido el usuario se llama a `AddStep()` o `AddWeight()` que se encargan de registrar el valor introducido por el usuario e insertarlo en la base de datos junto con la fecha actual (ver figura 7.60).

El funcionamiento del resto de los botones en esta vista es similar, se declara el listener que permanece activo a la espera de la ejecución de dicho evento.

```
private void AddStep() {
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "d-M-y");
    String today = dateFormat.format(new Date());
    AlertDialog.Builder builder = new AlertDialog.Builder( context: UserProfileActivity.this);
    final NumberPicker numberPicker = new NumberPicker( context: UserProfileActivity.this);
    numberPicker.setMaxValue(100000);
    numberPicker.setMinValue(0);
    builder.setTitle("Añadir Pasos");
    builder.setMessage("Inserta el número de pasos realizados en el día" + " " + today);
    builder.setView(numberPicker);

    builder.setPositiveButton( text: "OK", new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface dialog, int which) {
            int num = numberPicker.getValue();
            Date now = Calendar.getInstance().getTime();
            FirebaseUser currentUser = auth.getCurrentUser();
            //añadir en base de datos
            Steps toCreate = StepsService.getInstance().parseSteps(num);
            uploadStepsToFirebase(toCreate, currentUser.getUid());
        }
    });
    builder.setNegativeButton( text: "CANCEL", new DialogInterface.OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog, int which) {

        }
    });
    builder.create();
    builder.show();
}
```

Figura 7.60: Código del método AddStep()

El código de las secciones anteriormente mencionadas es el siguiente:

- **Información personal:** para mostrar la información del usuario, en el onCreate() de la actividad el siguiente código se encarga de hacer la consulta a la base de datos y llenar los campos (ver figura 7.61).

```

db.child("users").child(uid).addValueEventListener(new ValueEventListener()
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        setProfileImage();
        getSupportActionBar().setTitle("Mi perfil");

        User u = snapshot.getValue(User.class);
        name.setText(u.getName() + " " + u.getLastname());
        email.setText(u.getEmail());
        age.setText(u.getAge().toString() + " " + "años");
        date.setText("Miembro desde" + " " + u.getStart_date());

        diet_id = u.getDiet();
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Log.e(tag, "OnFailureUserProfile: ", error.toString());
    }
);

```

Figura 7.61: Consulta que obtiene la información personal de un usuario

El botón “Cambiar imagen“ abre la galería y cuando un usuario selecciona una imagen, esta se sube a Firebase Storage y se asocia a su perfil (ver figura 7.62).

```

private void getImageFromPhone(){
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    intent.putExtra(name: "crop", value: "true");
    intent.putExtra(name: "return-data", value: true);
    Intent pickIntent = new Intent(Intent.ACTION_PICK, android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    pickIntent.setType("image/*");

    Intent chooserIntent = Intent.createChooser(intent, "Selecciona una imagen");
    chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS, new Intent[] {pickIntent});

    startActivityForResult(chooserIntent, PICK_IMAGE);
}

```

Figura 7.62: Código que abre la galería del dispositivo

El botón “Gráfico de la dieta actual“ abre una nueva ventana que muestra el gráfico (ver figura 7.62).

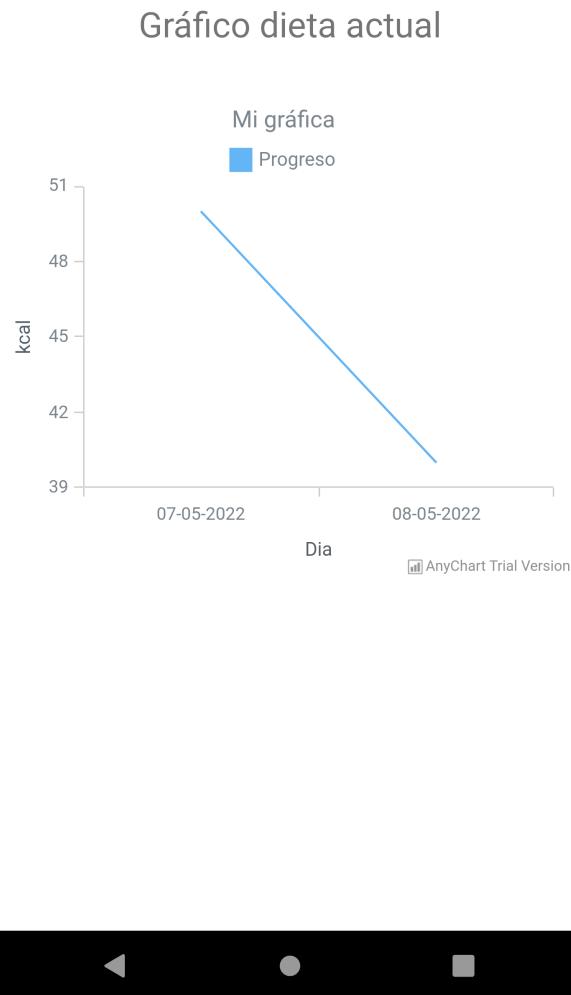


Figura 7.63: Vista de Gráfico de la dieta actual

- **Gráficas de pesos y pasos:** en esta sección el usuario dispone de un botón deslizable que activa las dos gráficas disponibles, la de pesos y la de pasos para que pueda visualizarlas. El botón en cuestión se compone de dos partes, un círculo (ver figura 7.65) que al pulsar se mueve de derecha a izquierda y viceversa con una animación de desplazamiento y un fondo (ver figura 7.66) que cambia de color blanco a verde y viceversa (ver figura 7.64).

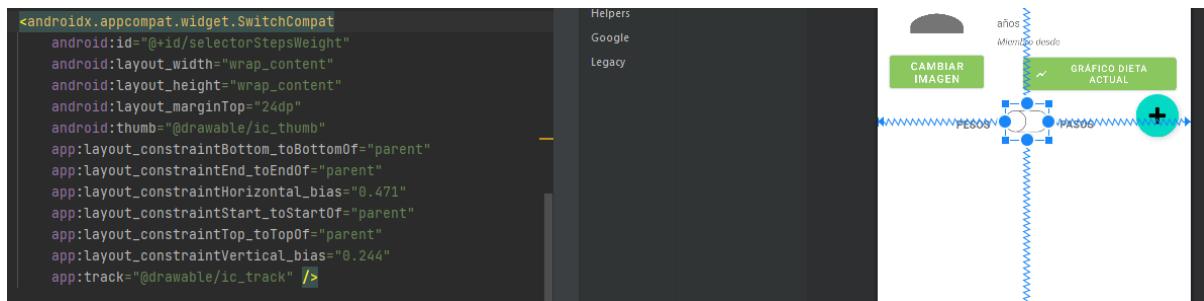


Figura 7.64: Layout del botón selector

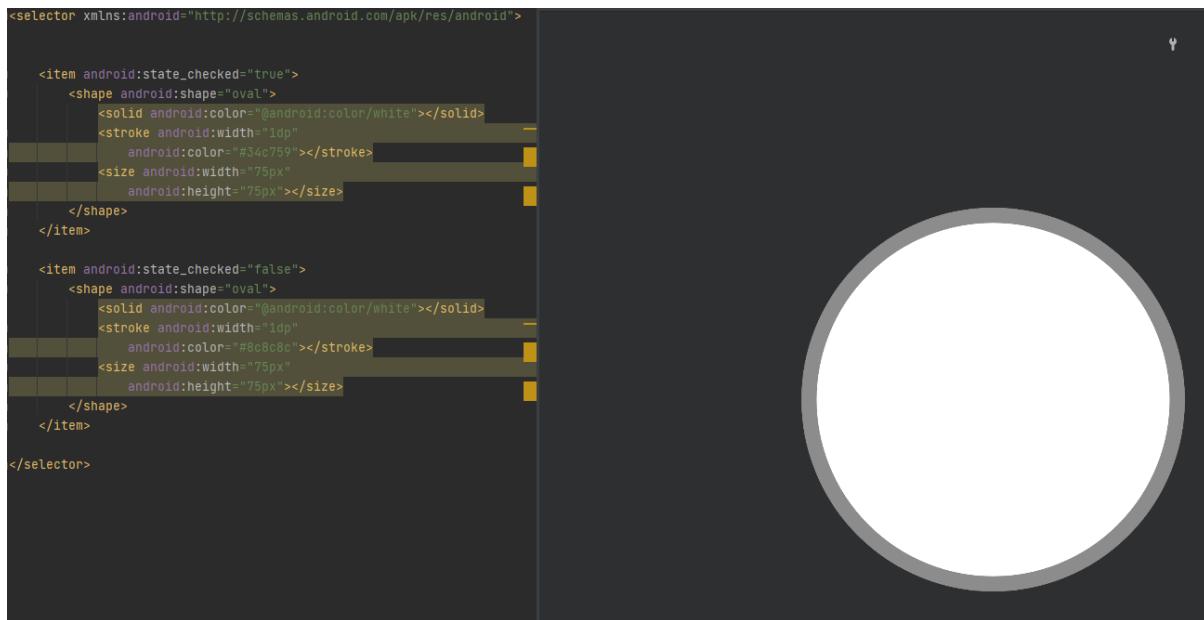


Figura 7.65: Diseño del thumb del botón selector

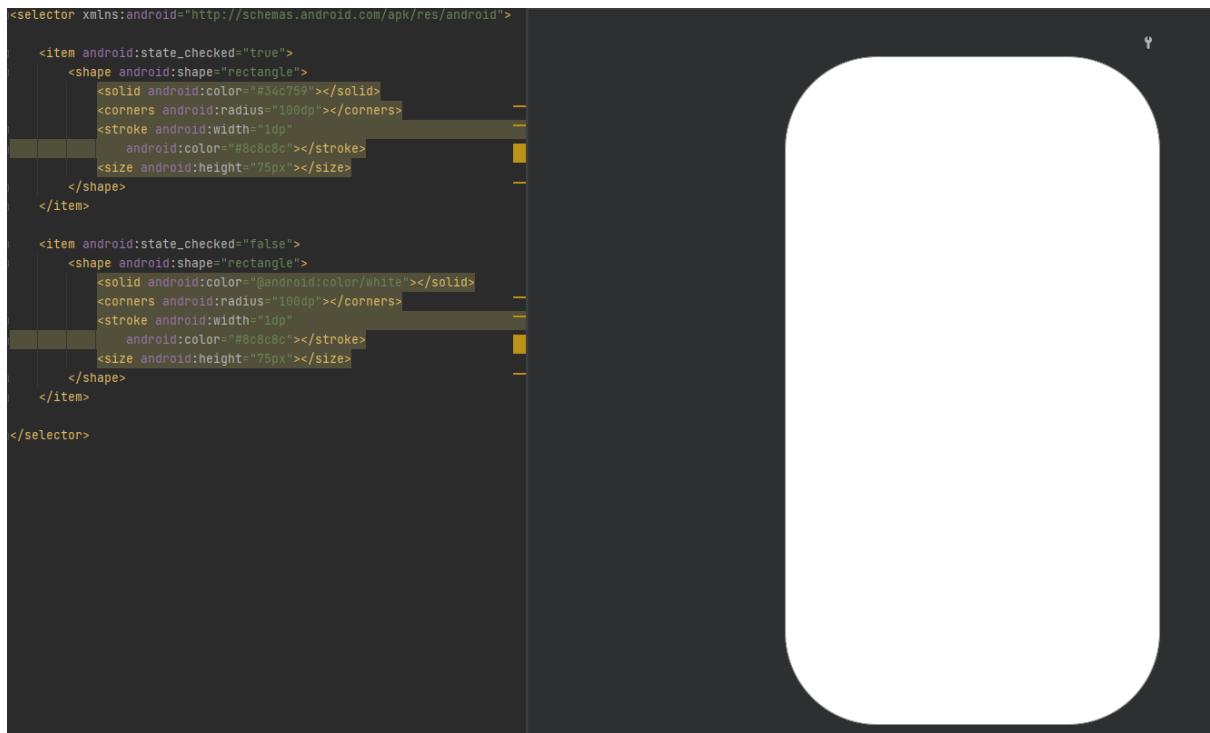


Figura 7.66: Diseño del track del botón selector

Al hacer clic sobre el botón se ejecuta el código asociado a él, que en función de si está a “true” o “false” activa una gráfica u otra (ver figura 7.67).

```

selectorStepsWeight.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if(isChecked){
            generateStepsChart();
            APIlib.getInstance().setActiveAnyChartView(steps);
        } else{
            generateWeightsChart();
            APIlib.getInstance().setActiveAnyChartView(weights);
        }
    }
});
selectorStepsWeight.setChecked(true);

```

Figura 7.67: Código del botón selector de gráfica

Para las gráficas de la aplicación se ha utilizado la librería de AnyChart [29]. Ambas gráficas tienen una implementación similar ya que su almacenamiento en la base de datos es similar de modo que lo único que cambia es la tabla donde están los datos pero el formato y la renderización de estos es idéntica. En la figura 7.68 se puede ver cómo está implementada la gráfica de pasos.

```

private void generateStepsChart(){
    steps = findViewById(R.id.dietchart);
    db.child("pasos").child(auth.getCurrentUser().getUid()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
        @Override
        public void onComplete(@NonNull Task<DataSnapshot> task) {
            Cartesian cartesian = AnyChart.line();
            cartesian.animation(true);

            cartesian.crosshair().enabled(true);
            cartesian.crosshair()
                .yLabel(true)
                // TODO ystroke
                .yStroke((Stroke) null, thickness: null, dashpattern: null, (String) null, (String) null);

            cartesian.tooltip().positionMode(TooltipPositionMode.POINT);

            cartesian.title("Mis pasos");

            cartesian.yAxis(index: 0).title("Pasos");
            cartesian.xAxis(index: 0).title("Dia");
            cartesian.xAxis(index: 0).labels().padding(5d, 5d, 5d, 5d);

            List<DataEntry> seriesData = new ArrayList<>();

            for(DataSnapshot ds : task.getResult().getChildren()){
                String pasos = ds.getValue().toString();

                String fecha = ds.getKey();
                seriesData.add(new CustomDataEntry(fecha, Integer.valueOf(pasos)));
            }

            Set set = Set.instantiate();
            set.data(seriesData);
            Mapping series1Mapping = set.mapAs(mapping: "{ x: 'x', value: 'value' }");

            Line series1 = cartesian.line(series1Mapping);
            series1.name("Progreso");
            series1.hovered().markers().enabled(true);
            series1.hovered().markers()
                .type(MarkerType.CIRCLE)
                .size(4d);
            series1.tooltip()
                .position("right")
                .anchor(Anchor.LEFT_CENTER)
                .offsetX(5d)
                .offsetY(5d);

            cartesian.legend().enabled(true);
            cartesian.legend().fontSize(13d);
            cartesian.legend().padding(0d, 0d, 10d, 0d);

            steps.setChart(cartesian);
            steps.setVisibility(View.VISIBLE);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.d(tag: "OnFailureUserProfile: ", msg: "");
            e.printStackTrace();
        }
    });
}
}

```

Figura 7.68: Código del gráfico de pasos

- **Acciones:** si el usuario presiona en el botón con 3 puntos verticales se le desplegará un menú con 4 opciones (ver figura 7.69), cada opción tiene asociada su función en el código (ver figura 7.70):

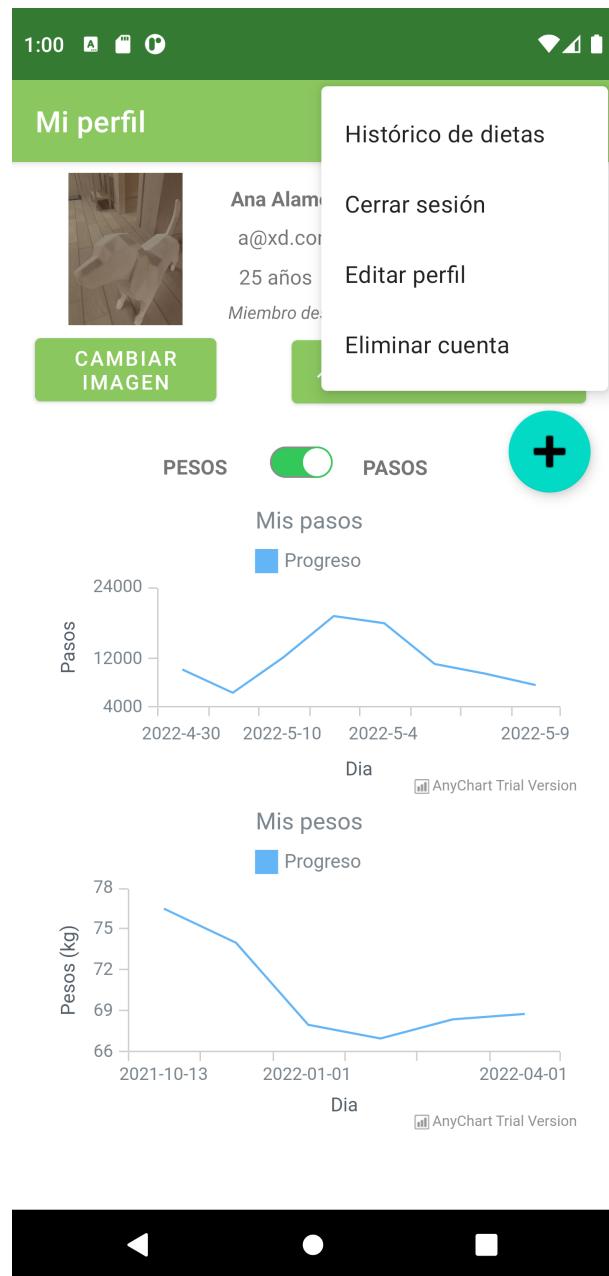


Figura 7.69: Vista “Mi Perfil“ con la lista de acciones desplegada

```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()){
        case R.id.DietHistory:
            DietHistory();
            break;
        case R.id.logoutMenu:
            logout();
            break;
        case R.id.editProfileMenu:
            editProfile();
            break;
        case R.id.deleteProfileMenu:
            deleteProfile();
            break;
    }
    return true;
}
```

Figura 7.70: Código de la lista de acciones

- **Historial de dietas:** al seleccionarlo se abre la vista Historial de dietas (ver figura 7.71). El método DietHistory() se encarga de obtener la lista de dietas de la base de datos y de mostrar la vista (ver figura 7.72).

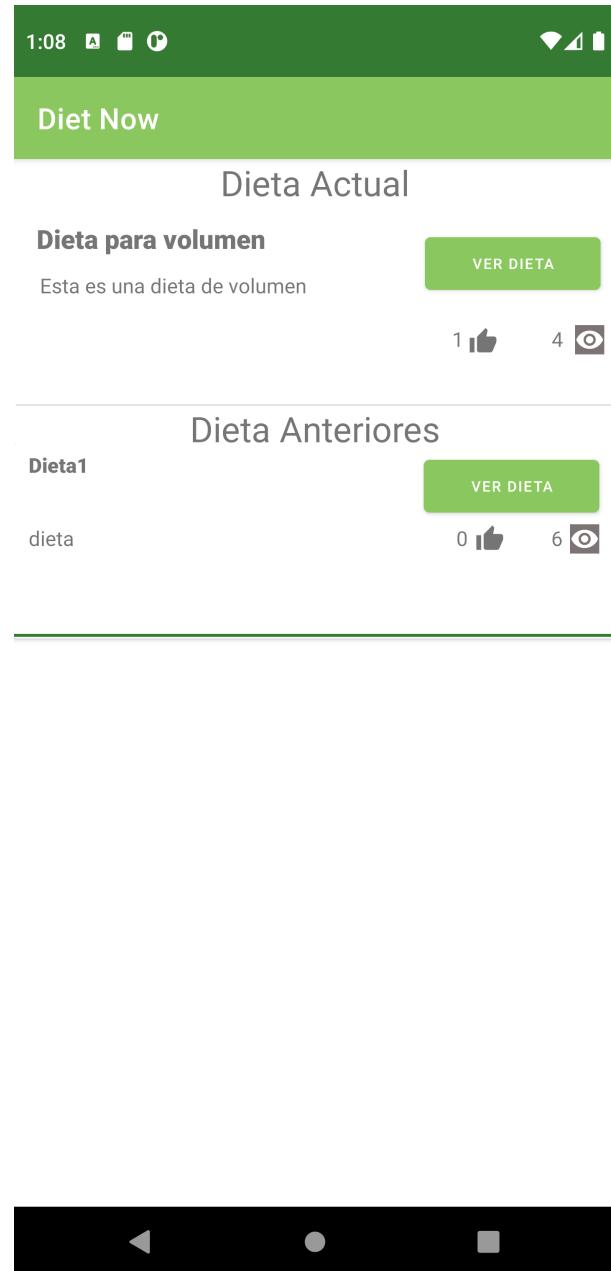


Figura 7.71: Vista Histórico de dietas

```

private void DietHistory(){
    Intent intent = new Intent( packageContext: UserProfileActivity.this, DietHistory.class);
    ArrayList<Diet> array = new ArrayList<>();
    db.child("diets").get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
        @Override
        public void onComplete(@NonNull Task<DataSnapshot> task) {
            for (DataSnapshot ds2 : task.getResult().getChildren()) {
                String titulo = ds2.child("title").getValue().toString();
                HashMap<String, Boolean> visit = ds2.child("visits").getValue(new GenericTypeIndicator<HashMap<String, Boolean>>(){});
                HashMap<String, Boolean> rating = ds2.child("rating").getValue(new GenericTypeIndicator<HashMap<String, Boolean>>(){});
                Boolean active = ds2.child("active").getValue(Boolean.class);
                String descripcion = ds2.child("description").getValue().toString();
                boolean published = ds2.child("published").getValue(Boolean.class);
                String id = ds2.getKey();
                Diet us = new Diet(descripcion, titulo, visit, rating);
                us.setId(ds2.child("id").getValue().toString());
                array.add(us);
            }
            intent.putExtra( name: "Dietas", array);
            startActivity(intent);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.d( tag: "OnFailureUserProfile: ", msg: "");
            e.printStackTrace();
        }
    });
}

```

Figura 7.72: Código del método `DietHistory()` que genera la vista Histórico de dietas

- **Cerrar sesión:** cierra la sesión del usuario y vuelve la pantalla de inicio de sesión (ver figura 7.73).

```

private void logout(){
    auth.signOut();
    /**
     * Eliminar todas las actividades anteriores y empezar una nueva
     * https://localcoder.org/android-kill-all-activities-on-logout
     */
    Intent intent = new Intent( packageContext: UserProfileActivity.this, MainActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK|Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(intent);
    finish();
}

```

Figura 7.73: Código del método `logout()` que genera cierra sesión y redirige al login

- **Editar perfil:** abre la vista de edición de perfil (ver figura 7.74). El método `editProfile()` se encarga de crear la nueva vista (ver figura 7.75).

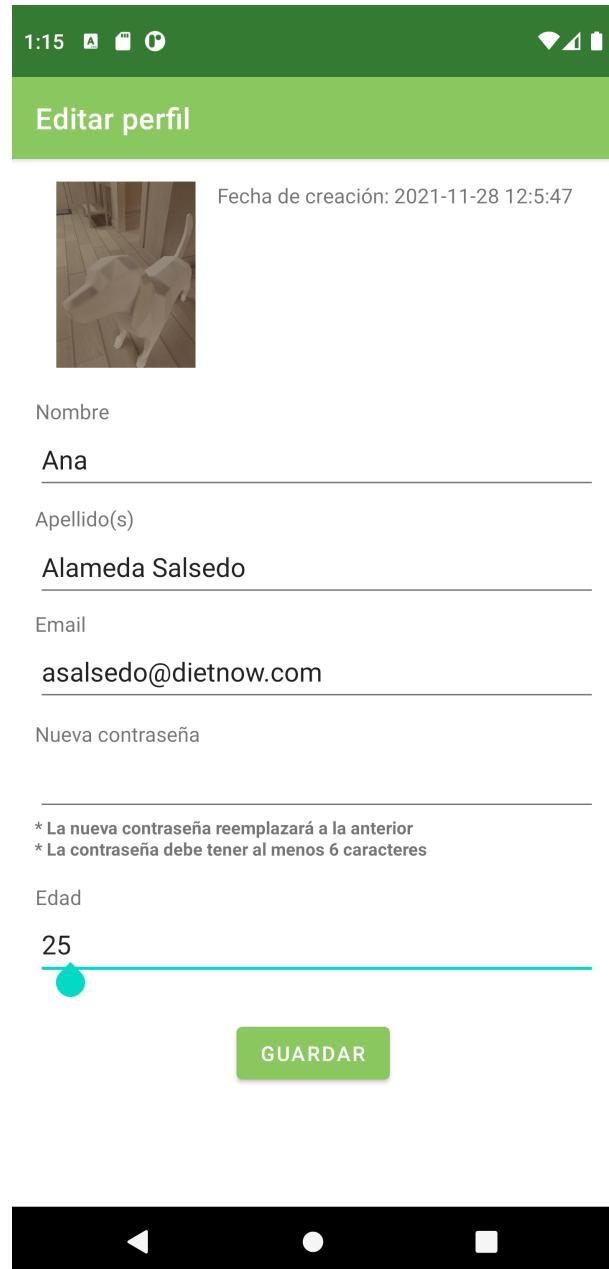


Figura 7.74: Vista de editar perfil

```
private void editProfile(){
    Intent intent = new Intent(packageContext: UserProfileActivity.this, UserProfileEditActivity.class);
    intent.putExtra(name: "uid", uid);
    startActivity(intent);
}
```

Figura 7.75: Código del método editProfile()

- **Eliminar perfil:** deshabilita el perfil en la base de datos y vuelve a la pantalla de inicio (ver figura 7.76).

```
private void deleteProfile() {
    FirebaseUser authUser = auth.getCurrentUser();

    if(authUser == null){
        Toast.makeText(getApplicationContext(),
            text: "El usuario no se ha borrado correctamente",
            Toast.LENGTH_LONG).show();
    } else{
        db.child("users").child(authUser.getUid()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
            @Override
            public void onComplete(@NonNull Task<DataSnapshot> task) {
                User user = task.getResult().getValue(User.class);
                user.setActive(false);
                Map<String, Object> userValues = user.toMap();
                db.child("users").child(authUser.getUid()).updateChildren(userValues);

                // cierra sesion y vuelve al login
                logout();
            }
        });
    }
}
```

Figura 7.76: Código del método `delete()` que deshabilita el perfil

Capítulo 8

Evaluación

En este capítulo se mostrarán los datos sobre las evaluaciones escritas por usuarios que han probado la aplicación. Se ha evaluado el rol del deportista mediante una encuesta condicionada según las funciones probadas por el propio usuario.

8.1. Diseño de la evaluación

Para realizar esta evaluación se ha ofrecido un fichero ejecutable (APK) a un conjunto de personas para probar la aplicación. Se ha ofrecido esta aplicación a un total de 23 personas, mayormente jóvenes, algunos que ya tenían experiencia con todo lo relacionado con la nutrición, dado que lo hacían ya de manera habitual en su vida cotidiana y otros que no tenían la misma experiencia en nutrición y por tanto era más probable que optasen por seguir una dieta.

La evaluación del usuario se ha realizado mediante un formulario de Google Forms [30] recogiendo, en varias secciones, los diferentes contenidos de la aplicación:

- Una primera sección con las diferentes preguntas personales y características generales de la aplicación, es decir, funciones que todo el mundo ha pasado por ellas como el registro, el menú principal y su perfil. Al final de esa sección se realizará una pregunta sobre la sección de creación de una dieta en la cual en caso afirmativo saltará a la sección correspondiente realizando las diferentes preguntas sobre la creación y gestión de la dieta. En caso negativo saltará a un bloque común que realizará también el otro usuario una vez termine la sección de creación de la dieta.
- El bloque de la gestión de la dieta se encarga de realizar las preguntas respectivas a todo el tema de creación de la dieta, inserción de alimentos y modificación de la misma. Una vez finalizado se realizará una pregunta (tanto si contestaste que sí como si comentaste que no en la pregunta que saltó a este bloque) se realizará una pregunta en la que se cuestionara si el usuario ha seguido alguna dieta en su transcurso por la aplicación. Esta respuesta, en caso de ser afirmativa te redirigirá a otra sección con preguntas respectivas a dicho módulo y en caso negativo te redirigirá a la sección final de la encuesta.
- El tercer bloque de seguimiento de la dieta hace referencia, a la parte del menú principal de “mi dieta“ con preguntas sobre la inserción de alimentos consumidos y

de comentarios principalmente. Una vez finalizada saltará a la sección final donde se dará una nota al proyecto.

- El último módulo de la encuesta se basa principalmente en una valoración general de la dieta.

8.2. Resultados de la evaluación

En la figura 8.1 se puede contemplar que casi un 62 % de los usuarios son del género masculino frente a un 38 % son del género femenino. Esto se debe a que la mayoría de los usuarios que resultan haber probado la aplicación son de la misma facultad.

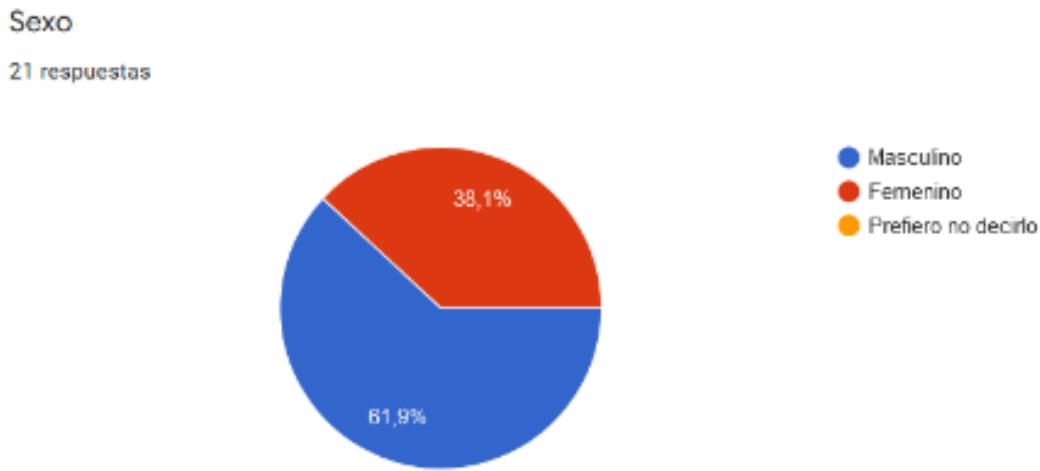


Figura 8.1: Resultado de la pregunta de género

En la figura 8.2 se puede deducir que ningún usuario tuvo ningún problema a la hora de registrarse.

¿has tenido alguna dificultad al registrarte?

23 respuestas

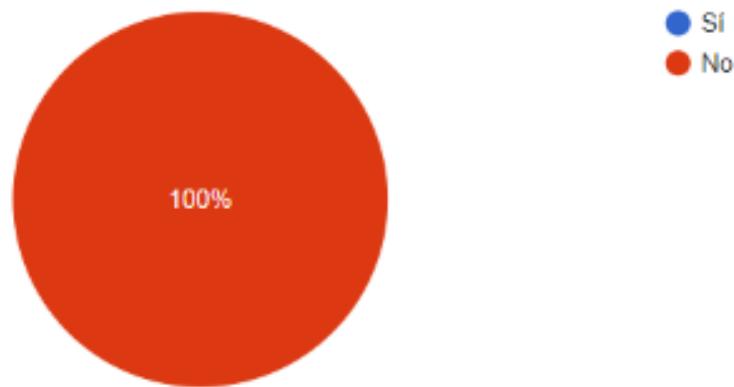


Figura 8.2: Resultado de la pregunta de la dificultad de registro

En la figura 8.3 se puede observar que casi un 74 % de los usuarios creen que el menú principal tiene las características necesarias para llevar a cabo la actividad de la aplicación frente a un 21,7 % que no saben a ciencia cierta si las tiene.

¿El menu principal ofrece las características necesarias?

23 respuestas

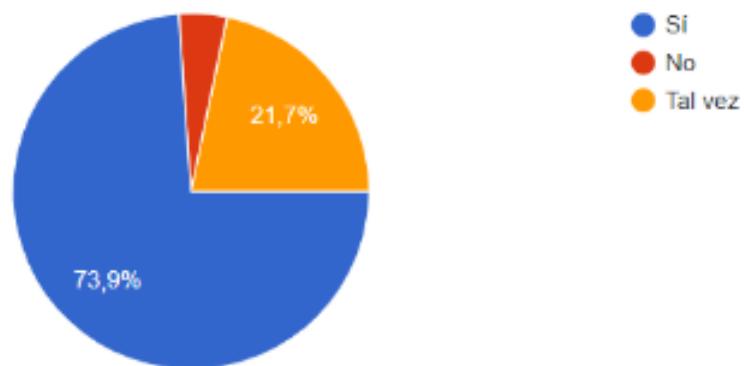


Figura 8.3: Resultado de la pregunta de página principal intuitiva

En la figura 8.4 se puede observar que nadie piensa que el perfil no cuente con la información necesaria para lo que está destinada la aplicación.

¿El perfil contiene la información necesaria para desarrollar la actividad?

23 respuestas

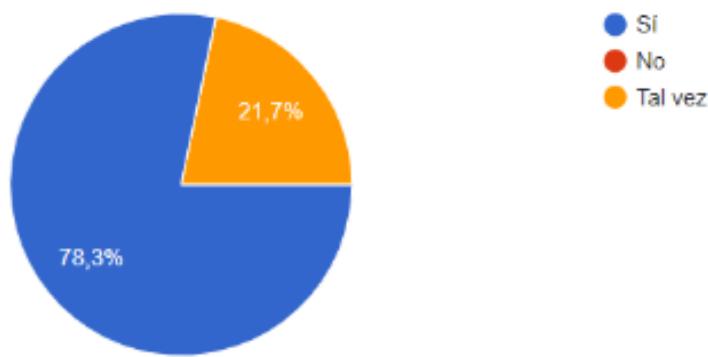


Figura 8.4: Resultado de la pregunta de información en el perfil

En la figura 8.5 se puede observar que un 65,2% de los usuarios que han probado la aplicación no han creado una dieta frente a un 34,8% de los usuarios si lo han hecho.

¿Has creado alguna dieta en la aplicación?

23 respuestas

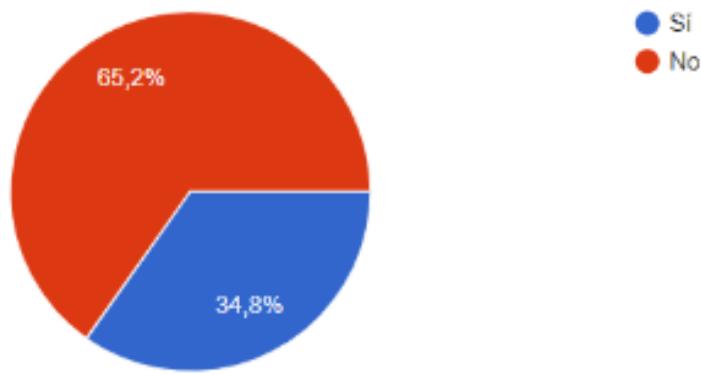


Figura 8.5: Resultado de la pregunta de creación de dietas

Como se puede observar en la figura 8.6 hay opiniones más diversas en cuanto a la complejidad de la creación de las dietas. Un 50% les ha parecido fácil la creación de la dieta frente a un 37,5% que les ha parecido compleja.

¿Te parece intuitiva la creación de la dieta?

8 respuestas

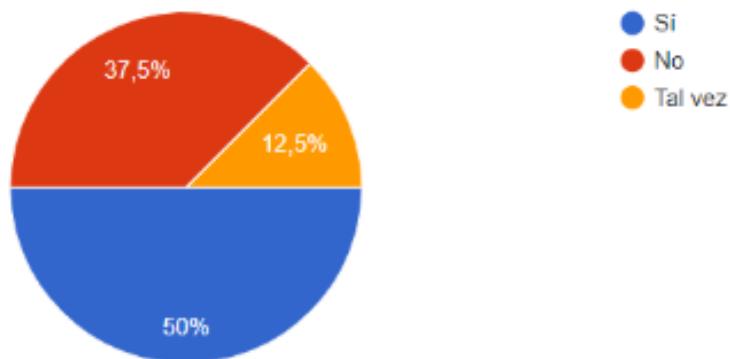


Figura 8.6: Resultado de la pregunta del proceso de creación de una dieta

En cuanto a la figura 8.7 se puede observar que solo un usuario respondió esta pregunta (se trata de una pregunta no obligatoria), opinando sobre una posible mejora respecto a la generación de una dieta.

¿Mejorarías algo de la creación de la dieta?

1 respuesta

Poder editar los datos en el momento de la inserción

Figura 8.7: Resultado de la pregunta de mejoras en la aplicación

Se puede observar en la figura 8.8 que un 73,9 % de los usuarios han seguido alguna dieta en el tiempo que estuvo probando la aplicación frente a un 26,1 % que no lo hizo.

¿Has seguido alguna dieta?

23 respuestas

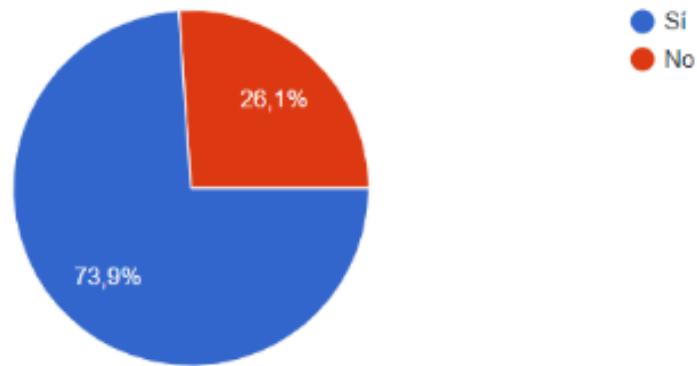


Figura 8.8: Resultado de la pregunta de seguimiento de una dieta

Se puede observar en la figura 8.9 que no ha habido ningún usuario que no le parezca intuitiva la inserción de alimentos.

¿Te parece intuitiva la pagina de insertar los alimentos correspondientes al dia?

17 respuestas

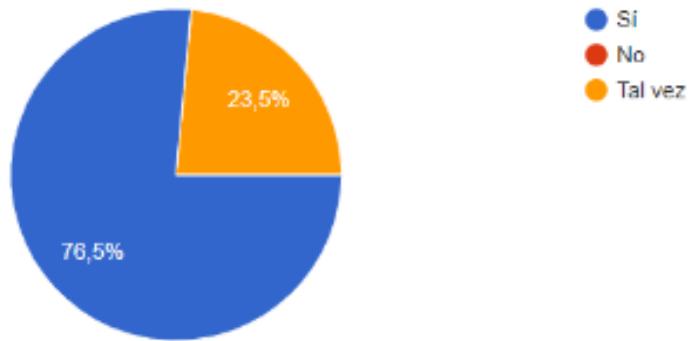


Figura 8.9: Resultado de la pregunta de inserción de alimentos en una dieta

En la figura 8.10 se puede observar que un 58,8 % de los usuarios encuentran utilidad a la sección de comentarios de la dieta frente a un 11,8 % que no lo encuentran.

¿Te parece útil la sección de comentarios de la dieta?

17 respuestas

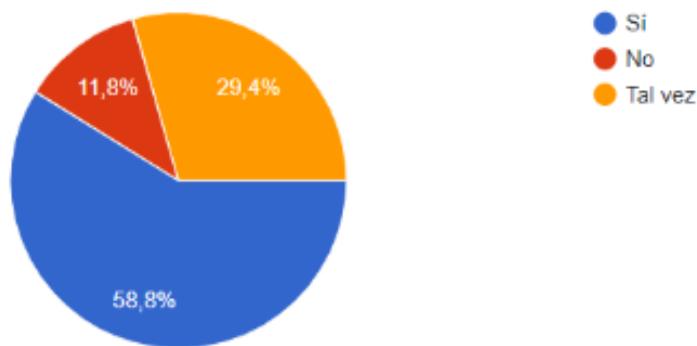


Figura 8.10: Resultado de la pregunta de utilidad de los comentarios

En la figura 8.11 se encuentra un comentario de una persona ofreciendo una posible mejora a la aplicación en la sección de seguimiento de la dieta.

¿Mejorarías algo de la sección de la dieta seguida?

1 respuesta

Al pulsar el alimento que te salgs la información nutricional

Figura 8.11: Resultado de la pregunta de mejoras en seguimiento de la dieta

Como se puede observar en la figura 8.12 la valoración general de la aplicación tiene una media de 7,52 en unas 23 respuestas insertadas por los usuarios.

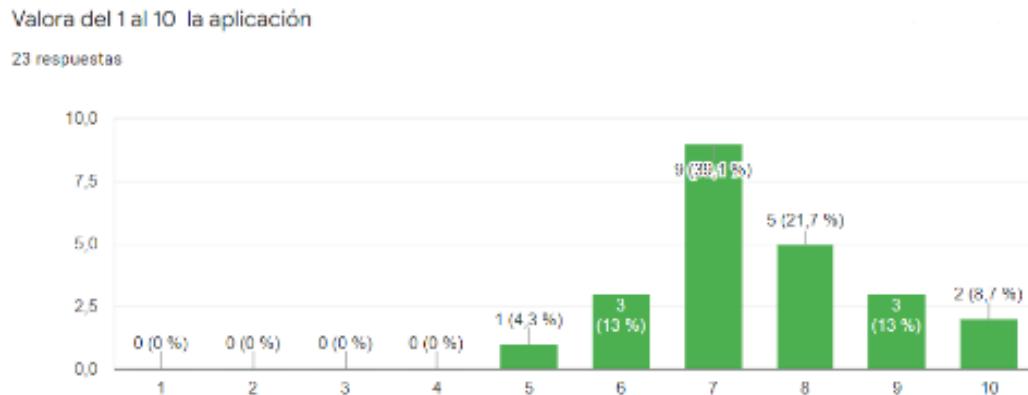


Figura 8.12: Resultado de la pregunta de valoración general de la aplicación

Respecto a la figura 8.13 se puede observar un comentario de una persona en la que valora una posible mejora para la aplicación.

¿Te gustaría que se añadiese alguna función extra a la aplicación?

1 respuesta

Que las gráficas se pudiesen modificar (ejemplo, en vez de por días, por meses haciendo una media)

Figura 8.13: Resultado de la pregunta de funcionalidades extra

Capítulo 9

Conclusiones y trabajo futuro

9.1. Conclusiones

En este proyecto se ha realizado una aplicación que ayuda a gestionar la dieta de un deportista, pudiendo elegir una dieta entre las creadas por otros usuarios.

Los deportistas pueden crear dietas que posteriormente pueden ser seguidas por otros usuarios. Además los deportistas, tienen la posibilidad de consultar la información detallada de cada uno de los alimentos que deben de consumir durante la realización de la dieta. A su vez, se pueden subir documentos explicativos para que el deportista que sigue la dieta conozca la finalidad cada alimento dentro de la dieta, pudiendo aportar mayor información al usuario.

Se puede valorar la dieta actual para que otros deportistas tenga referencias de ella, pudiendo a su vez, hacer comentarios sobre la dieta seguida.

En el siguiente enlace se puede ver y descargar, desde el repositorio de GitHub, el código del proyecto así como la aplicación ejecutable: <https://github.com/csegundo/Diet-Now>.

9.2. Trabajo futuro

A continuación se van a describir una serie de ideas de trabajo futuro que se podrían añadir a la aplicación con el objetivo de mejorar la experiencia del usuario durante el uso de la misma.

- **Barra inferior de navegación:** el objetivo principal de esta implementación es la mejora de la experiencia del usuario, permitiéndole una navegación más sencilla e intuitiva entre las diferentes vistas añadiendo en la zona inferior de la pantalla un menú con una serie de iconos.
- **Inicio de sesión con una cuenta de Google o Facebook:** para mejorar la usabilidad de la aplicación, se podría implementar esta nueva funcionalidad de inicio de sesión con una cuenta de Google o Facebook. Además del beneficio ya mencionado, podría aumentar considerablemente el número de usuarios en la aplicación debido

a que la mayoría de usuarios de Facebook utilizan un dispositivo móvil, como se muestra en la Figura 9.1.

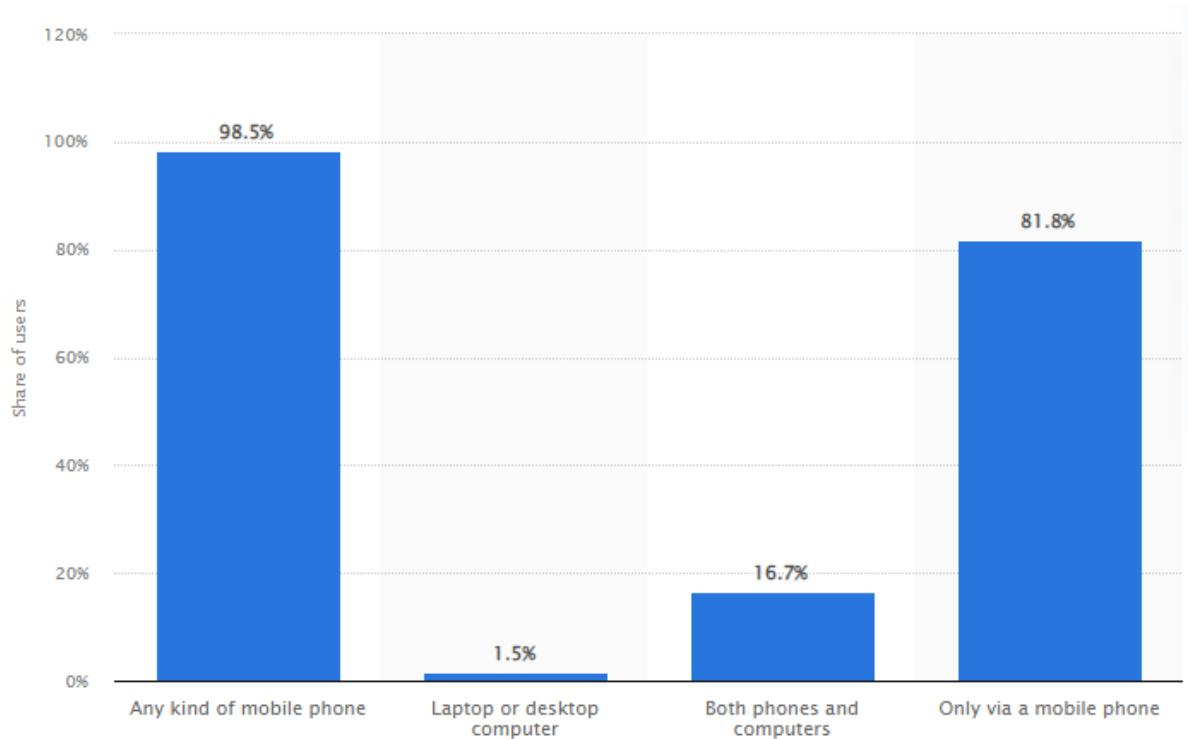


Figura 9.1: Dispositivos utilizados para iniciar sesión en Facebook en Enero de 2022 [1]

- **Restablecer contraseña:** actualmente la aplicación no dispone de un sistema de recuperación de contraseña en el caso de que el usuario no recuerde la misma. Esta implementación facilitaría al usuario recuperar el acceso a su cuenta cuando olvida la contraseña, añadiendo un nuevo botón en la pantalla de inicio de sesión que envía un correo electrónico al email del usuario para que éste pueda restablecer la contraseña desde un enlace.
- **Añadir alimentos en grupo de comidas:** dar la posibilidad al deportista de añadir alimentos a una dieta en cada una de las cinco comidas que se realizan a lo largo del día. Cuando se va a añadir un alimento a una dieta, el usuario especifica si ese alimento pertenece al desayuno, comida, merienda...
- **Compartir dietas:** permitir a los usuarios que usen la aplicación la posibilidad de compartir mediante otras redes sociales o de mensajería las diferentes dietas insertadas en la aplicación.
- **Conexión con pulseras inteligentes:** ofrecer la posibilidad al usuario de conectar una pulsera o reloj inteligente a la aplicación. Mediante esta conexión, se pueden recopilar datos como el número de pasos caminados durante el día e introducirlos de forma automática en la cuenta del usuario para que se muestren en la gráfica de los pasos.
- **Conectar la gestión de dietas con asistente Alexa o Google:** permitir a los usuarios que deseen crear una dieta la posibilidad de hacerlo mediante voz.

Esta herramienta se podría utilizar también para generar una compra mediante el asistente de Alexa o Google con los productos almacenados de una dieta.

Chapter 9

Conclusions and future work

9.1. Conclusions

In this project we have developed an application that helps to manage the diet of an athlete, being able to choose a diet among those created by other users.

Athletes can create diets that can later be followed by other users. In addition, athletes have the possibility to consult the detailed information of each of the foods that they should consume during the diet. At the same time, explanatory documents can be uploaded so that the athlete who follows the diet knows the purpose of each food within the diet, being able to provide more information to the user.

The current diet can be evaluated so that other athletes have references of it, being able, in turn, to comment on the diet followed.

In the following link you can view and download, from the GitHub repository, the code of the project as well as the executable application: <https://github.com/csegundo/Diet-Now>.

9.2. Future work

The following are a number of ideas for future work that could be added to the application in order to improve the user's experience while using the application.

- **Bottom navigation bar:** the main objective of this implementation is to improve the user experience, allowing an easier and more intuitive navigation between the different views by adding a menu with a series of icons at the bottom of the screen.
- **Signing in with a Google or Facebook account:** to improve the usability of the application, this new functionality of logging in with a Google or Facebook account could be implemented. In addition to the benefit already mentioned, it could considerably increase the number of users in the application due to the fact that most Facebook users use a mobile device, as shown in Fig. 9.1.

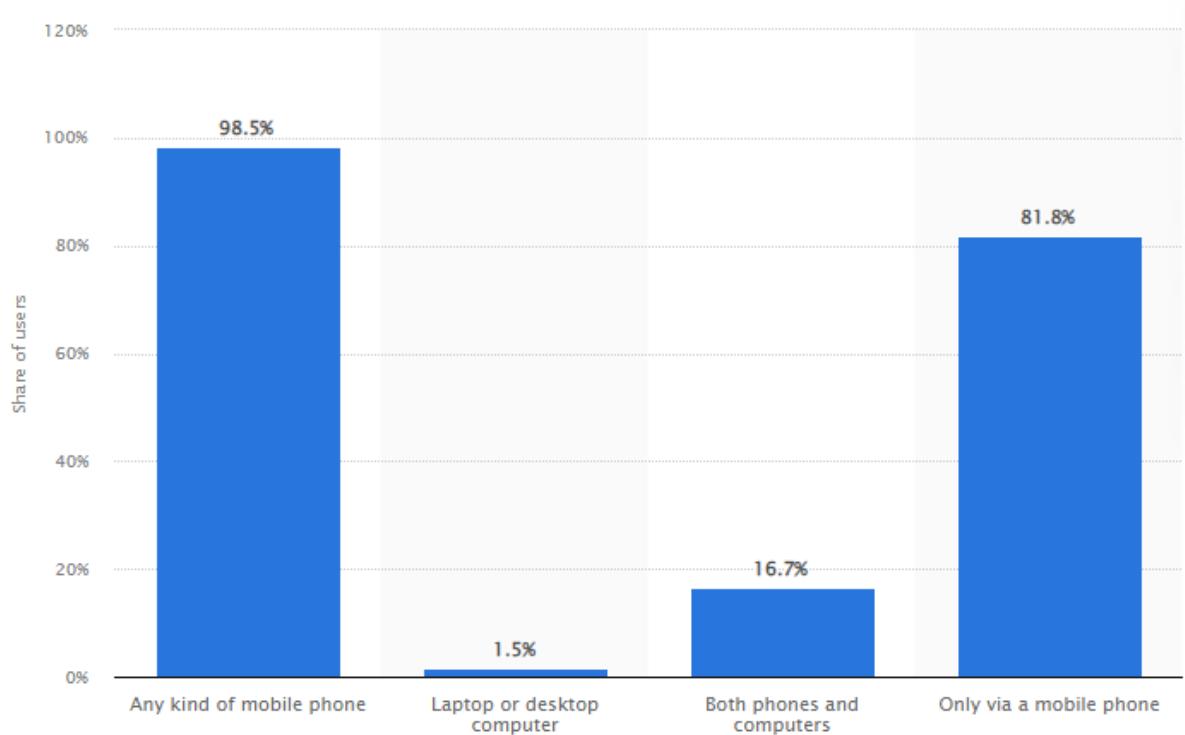


Figura 9.1: Devices used to log in to Facebook in January 2022 [1]

- **Reset password:** currently the application does not have a password recovery system in case the user does not remember the password. This implementation would make it easier for the user to regain access to their account when they forget their password by adding a new button on the login screen that sends an email to the user's email so that the user can reset the password from a link.
- **Adding foods in food groups:** give the athlete the possibility to add food to a diet in each of the five meals that are eaten throughout the day. When a food is to be added to a diet, the user specifies whether that food belongs to breakfast, lunch, afternoon snack...
- **Share diets:** allow users who use the application the possibility of sharing through other social or messaging networks all different diets they inserted into the application.
- **Connection with smart watches:** offer the possibility for the user to connect a wristband or smartwatch to the application. Through this connection, data such as the number of steps walked during the day can be collected and automatically entered into the user's account to be displayed on the step graph.
- **Connect diet management with Alexa or Google assistant:** allow users who create a diet the possibility of doing so by voice commands. This tool could also be used to generate a purchase through the Alexa or Google assistant with the stored products of a diet.

Capítulo 10

Aportaciones individuales

En este capítulo se listan las diferentes aportaciones realizadas por cada miembro de los participantes del proyecto. Aunque haya módulos que se hayan realizado de manera más individual o en grupos de dos, para un mayor entendimiento, la mayoría de módulos desarrollados de han realizado de manera colectiva entre todos los miembros del proyecto así como la parte de testeo y resolución de *bugs* de la aplicación.

10.1. David Fernández Alejo

- **Ingeniería de requisitos:** al ser una parte esencial, para un mayor entendimiento del proyecto, participó junto a sus compañeros de manera activa en las diferentes lluvia de ideas propuestas por los diferentes integrantes del equipo. Además, Al igual que sus compañeros redactó y analizó los diferentes casos de uso que se iban a implementar en el proyecto.
- **Desarrollo:**
 - **Modelo de datos:** participó junto a sus compañeros en la elaboración del diagrama entidad relación de la base de datos para satisfacer los requisitos de la aplicación.
 - **Módulo de autenticación del usuario:** participó activamente en la creación de dicho modulo. Principalmente se encargo de la creación de los usuarios.
 - **Módulo de dieta actual:** participó activamente con la creación de dicho módulo. Principalmente para este módulo se encargó de gestionar los comentarios de una dieta, así como la gestión de las calorías consumidas durante la semana, introduciendo la lógica y el comportamiento de la interfaz.
 - **Módulo de gestión de usuarios:** participó activamente en la creación de dicho módulo. Principalmente fue el encargado del caso de uso de eliminar perfil de un deportista situado en el menú de los ajustes del perfil.
 - **Módulo de dietas:** participó activamente en la creación de dicho módulo.
 - **Módulo de todas las dietas:** participó activamente en la creación de dicho módulo. Su participación más característica a la hora de desarrollar

el modulo fue la creación del caso de uso encargado de suscribirse a una dieta.

- **Módulo de gestión de dieta:** participó activamente en la creación de dicho módulo. Principalmente fue el encargado de modificar y eliminar una dieta así como insertar alimentos dentro de la dieta.
- **Desarrollo de API REST en Node.js:** participó de forma activa en el desarrollo de esta API REST en Node.js para su uso en la aplicación.
- **Desarrollo de API REST en Spring:** participó de forma activa en el desarrollo de esta API REST en Spring Boot para su uso en la aplicación.
- **Memoria:** participó en la elaboración de esta memoria final así como en su creación en L^AT_EX. Se encargó de crear los capítulos tres, cuatro y siete así como los anexos.

10.2. Carlos Segundo Nieto

- **Ingeniería de requisitos:** al ser una parte esencial, para un mayor entendimiento del proyecto, participó junto a sus compañeros de manera activa en las diferentes lluvia de ideas propuestas por los diferentes integrantes del equipo. Además, Al igual que sus compañeros redactó y analizó los diferentes casos de uso que se iban a implementar en el proyecto.
- **Desarrollo:**
 - **Modelo de datos:** participó junto a sus compañeros en la elaboración del diagrama entidad relación de la base de datos para satisfacer los requisitos de la aplicación.
 - **Módulo de autenticación del usuario:** participó activamente en la creación de dicho modulo. Principalmente se encargo de la creación y la autenticación de los deportistas vía API.
 - **Módulo de dieta actual:** participó activamente con la creación de dicho módulo. Principalmente para este módulo se encargó de gestionar los comentarios de una dieta, así como la valoración en forma de “Me gusta” y “No me gustas”, al igual que las visitas. A su vez, se encargo de la gestión de las calorías consumidas durante la semana, introduciendo la lógica y el comportamiento de la interfaz
 - **Módulo de gestión de usuarios:** participó activamente en la creación de dicho módulo así como el cambio de los credenciales de autenticación de los diferentes usuarios. También fue el encargado de maquetar el perfil del administrador proporcionándole información útil acerca de su actividad.
 - **Módulo de dietas:** participó activamente en la creación de dicho módulo.
 - **Módulo de todas las dietas:** participó activamente en la creación de dicho módulo. Su participación más característica a la hora de desarrollar el módulo fue la del cambios de estado de publicación de la dieta deseada por el usuario así como la gestión de errores de la misma. También se

encargó principalmente de la vista correspondiente a visualizar los detalles de la dieta.

- **Módulo de gestión de dieta:** participó activamente en la creación de dicho módulo. Principalmente fue el encargado de modificar, crear y eliminar una dieta así como insertar alimentos dentro de la dieta. Otro de sus subprocessos principales para este modulo es la característica de descargar el documento insertado por el usuario que creo la dieta.
- **Desarrollo de API REST en Node.js:** participó de forma activa en el desarrollo de esta API REST en Node.js para su uso en la aplicación.
- **Desarrollo de API REST en Spring:** participó de forma activa en el desarrollo de esta API REST en Spring Boot para su uso en la aplicación. Fue el encargado de la integridad de datos enviados a los diferentes *endpoints* realizados en esta API para administradores.
- **Memoria:** participó en la elaboración de esta memoria final así como en su creación en L^AT_EX. Se encargó de crear los capítulos tres, cuatro y nueve así como los anexos.

10.3. Vitaliy Savchenko

- **Ingeniería de requisitos:** al ser una parte esencial, para un mayor entendimiento del proyecto, participó junto a sus compañeros de manera activa en las diferentes lluvia de ideas propuestas por los diferentes integrantes del equipo. Además, Al igual que sus compañeros redactó y analizó los diferentes casos de uso que se iban a implementar en el proyecto. Diseñó y creó los *mockups* iniciales de la aplicación.
- **Desarrollo:**
 - **Modelo de datos:** participó junto a sus compañeros en la elaboración del diagrama entidad relación de la base de datos para satisfacer los requisitos de la aplicación.
 - **Módulo de autenticación del usuario:** participó activamente en la creación de dicho modulo y ayudó en la búsqueda de información para los diferentes métodos de autenticación y creación de un usuario.
 - **Módulo de dieta actual:** participó activamente con la creación de dicho módulo aportando diferentes estilos para la creación de las diferentes vistas asociadas al modulo.
 - **Módulo de gestión de usuarios:** participó activamente en la creación de dicho módulo. Fue el encargado de la funcionalidad de eliminar un deportista de la plataforma así como su gestión de errores y alertas. También fue el encargado de la creación de las diferentes gráficas existentes en el perfil del usuario.
 - **Módulo de dietas:** participó activamente en la creación de dicho módulo.
 - **Módulo de todas las dietas:** participó activamente en la creación de dicho módulo. Fue el encargado de la creación de un *popup* con la información nutricional detallada de un producto obtenida vía API.

- **Módulo de gestión de dieta:** participó activamente en la creación de dicho módulo. Fue el responsable de la inserción de los alimentos ya sea vía API o vía manual insertando los datos en el formulario correspondiente. Otro proceso que realizó fue el correcto funcionamiento de la cámara como la gestión de los permisos Android para la implementación de un escáner encargado de detectar los códigos de barras de los alimentos y hacer la consulta a la API. Por ultimo implementó la funcionalidad de borrar un documento subido de una dieta.
- **Desarrollo de API REST en Node.js:** participó de forma activa en el desarrollo de esta API REST en Node.js para su uso en la aplicación.
- **Desarrollo de API REST en Spring:** participó de forma activa en el desarrollo de esta API REST en Spring Boot para su uso en la aplicación.
- **Memoria:** participó en la elaboración de esta memoria final así como en su creación en LATEX. Se encargó de crear los capítulos tres, cuatro y siete así como los anexos.

10.4. Víctor Velasco Arjona

- **Ingeniería de requisitos:** al ser una parte esencial, para un mayor entendimiento del proyecto, participó junto a sus compañeros de manera activa en las diferentes lluvia de ideas propuestas por los diferentes integrantes del equipo. Además, Al igual que sus compañeros redactó y analizó los diferentes casos de uso que se iban a implementar en el proyecto.
- **Desarrollo:**
 - **Modelo de datos:** participó junto a sus compañeros en la elaboración del diagrama entidad relación de la base de datos para satisfacer los requisitos de la aplicación.
 - **Módulo de autenticación del usuario:** participó activamente en la creación de dicho modulo aportando diferentes ideas y documentación para el correcto funcionamiento del mismo.
 - **Módulo de dieta actual:** participó activamente en la creación de dicho módulo. Aportó la funcionalidad de la posibilidad de dejar de seguir una dieta así como la gestión de errores del mismo. Otro de los desarrollos implementados por el integrante es la capacidad de insertar los alimentos consumidos por el usuario en concreto a la base de datos para su futuro conteo.
 - **Módulo de gestión de usuarios:** participó activamente en la creación de dicho módulo. Fue el encargado de la realización del módulo y gestión de los pesos y pasos introducidos diariamente por el usuario y su seguimiento. También se encargó de la creación de las diferentes vistas del menú principal, tanto del deportista como del administrador y así como la creación de una vista específica para el rol de administrador encargada de la visualización de todos los usuarios existentes en la plataforma.
 - **Módulo de dietas:** participó activamente en la creación de dicho módulo.

- **Módulo de todas las dietas:** participó activamente en la creación de dicho módulo. Su principal desarrollo en este apartado es la creación de una vista encargada de la visualización de todas las dietas creadas por el usuario que está autenticado así como la vista encargada de ver todas las dietas insertadas en la aplicación de manera dinámica. A su vez, también se encarga de la creación de un historial de las dietas que sigue el usuario situado en el menú de ajustes situado en el perfil del usuario.
- **Módulo de gestión de dieta:** participó activamente en la creación de dicho módulo. Fue uno de los responsables encargados de la modificación de los diferentes atributos del alimento insertado previamente en la dieta.
- **Desarrollo de API REST en Node.js:** participó de forma activa en el desarrollo de esta API REST en Node.js para su uso en la aplicación.
- **Desarrollo de API REST en Spring:** participó de forma activa en el desarrollo de esta API REST en Spring Boot para su uso en la aplicación.
- **Memoria:** participó en la elaboración de esta memoria final así como en su creación en L^AT_EX. Se encargó de crear los capítulos cuatro, seis y ocho así como los anexos.

Bibliografía

- [1] Statista. <https://www.statista.com/statistics/377808/distribution-of-facebook-users-by-device/>. 2022.
- [2] Indya. <https://getindya.com/>. 2022.
- [3] Oorenji. <https://oorenji.com/>. 2022.
- [4] Fat Secret. <https://www.fatsecret.es/>. 2022.
- [5] Nootric App. <https://www.nootric.com/es>. 2022.
- [6] Fitia. <https://fitia.app/>. 2022.
- [7] Android Studio. <https://developer.android.com/studio>. 2022.
- [8] Google Firebase. <https://firebase.google.com/>. 2022.
- [9] Google Firebase Authentication. <https://firebase.google.com/docs/auth>. 2022.
- [10] Google Firebase Realtime Database. <https://firebase.google.com/docs/database>. 2022.
- [11] Google Firebase Realtime Database. <https://firebase.google.com/docs/storage>. 2022.
- [12] Spring Boot. <https://spring.io/>. 2022.
- [13] Node.js. <https://nodejs.org/es/>. 2022.
- [14] Open Food Facts Android. <https://github.com/openfoodfacts/openfoodfacts-androidapp>. 2022.
- [15] Retrofit. <https://square.github.io/retrofit/>. 2022.
- [16] Git. <https://git-scm.com/>. 2022.
- [17] Trello. <https://trello.com/es>. 2022.
- [18] Overleaf. <https://es.overleaf.com/>. 2022.
- [19] David Pacios Izquierdo. <https://www.ucm.es/data/cont/docs/1346-2019-04-12-BaSix%20LaTeX%20ba%C81sico%20con%20ejercicios%20resueltos27.pdf>. 2022.
- [20] Google Drive. https://www.google.com/intl/es_es/drive/. 2022.
- [21] Visual Studio Code. <https://code.visualstudio.com/>. 2022.

- [22] Design patterns. <https://www.raywenderlich.com/18409174-common-design-patterns-and-app-architectures-for-android>. 2022.
- [23] JSON. <https://www.json.org/json-es.html>. 2022.
- [24] Jeremyh. <https://github.com/jeremyh/jBCrypt>. 2022.
- [25] Stat Counter. <https://gs.statcounter.com/os-market-share/mobile/europe>. 2022.
- [26] Android Developers. [https://developer.android.com/reference/android/app/Activity#finish\(\)](https://developer.android.com/reference/android/app/Activity#finish()). 2022.
- [27] Android Developers. <https://developer.android.com/studio/run/emulator-networking#networkaddresses>. 2022.
- [28] Yuriy Budiyev. <https://github.com/yuriy-budiyev/code-scanner>. 2022.
- [29] AnyChart. <https://www.anychart.com/es/>. 2022.
- [30] google. <https://www.google.es/intl/es/forms/about/>. 2022.
- [31] Open Food facts. <https://es.openfoodfacts.org/>. 2022.
- [32] AnyChart Android. <https://github.com/AnyChart/AnyChart-Android>. 2022.
- [33] Medium. <https://medium.com/javarevisited/a-simple-user-authentication-api-made-with-spring-boot-533a2a2a2a2a>. 2022.
- [34] Medium. <https://medium.com/@sukhbirsekhon3939/how-to-create-a-login-application-on-android-studio-d664662578f8>. 2022.
- [35] Freecodecamp. <https://www.freecodecamp.org/espanol/news/guia-para-principiantes-de-git-y-github/>. 2022.
- [36] Techigness. <https://www.techigness.com/post/how-to-draw-no-sql-data-model-diagram/>. 2022.
- [37] Openwebinars. <https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/>. 2022.
- [38] Firebase. <https://firebase.google.com/docs/firestore/manage-data/transactions>. 2022.
- [39] Java2s. http://www.java2s.com/Tutorials/Java/Data_Type_How_to/Date/Get_day_of_week_int_value_and_String_value.htm. 2022.
- [40] Android. <https://developer.android.com/reference/android/content/SharedPreferences>. 2022.
- [41] Stackoverflow. <https://stackoverflow.com/q/48307610>. 2022.
- [42] Spring. <https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html>. 2022.
- [43] Firebase. <https://firebase.google.com/docs/database/android/read-and-write?hl=es>. 2022.
- [44] Stackoverflow. <https://stackoverflow.com/a/34487195>. 2022.

- [45] Stackoverflow. <https://stackoverflow.com/q/64538836>. 2022.
- [46] Baeldung. <https://www.baeldung.com/sha-256-hashing-java>. 2022.
- [47] Google Firebase. <https://firebase.google.com/docs/database/android/structure-data?authuser=0>. 2022.
- [48] Stackoverflow. <https://stackoverflow.com/q/51277122>. 2022.
- [49] JWT. <https://jwt.io/introduction>. 2022.
- [50] Firebase. <https://firebase.google.com/docs/database/android/offline-capabilities>. 2022.
- [51] Developer Android. <https://developer.android.com/guide/topics/manifest/activity-element?hl=es-419>. 2022.
- [52] Android Developers. <https://developer.android.com/guide/topics/ui/floating-action-button?hl=es>. 2022.
- [53] Android. <https://developer.android.com/guide/components/activities/tasks-and-back-stack?hl=es-419>. 2022.
- [54] Google Firebase. <https://firebase.google.com/docs/database/admin/save-data>. 2022.
- [55] Code Palace. <https://youtu.be/drH63NpSWyk>. 2022.
- [56] Android. <https://developer.android.com/reference/android/app/Activity>. 2022.
- [57] Stackoverflow. <https://stackoverflow.com/q/31792072>. 2022.
- [58] Google Firebase. <https://firebase.google.com/docs/admin/setup>. 2022.
- [59] Jarroba. <https://jarroba.com/json-javascript-ejemplos/>. 2022.
- [60] Google Firebase. <https://firebase.google.com/docs/reference/android/com/google/firebase/auth/AuthCredential>. 2022.
- [61] Node.js. <https://nodejs.dev/learn/making-http-requests-with-nodejs>. 2022.
- [62] Google Firebase. <https://firebase.google.com/docs/auth/android/manage-users>. 2022.
- [63] Tutorialspoint. https://www.tutorialspoint.com/nodejs/nodejs_first_application.htm. 2022.
- [64] Google Firebase. <https://firebase.google.com/docs/auth/users>. 2022.
- [65] Geeksforgeeks. <https://www.geeksforgeeks.org/sha-256-hash-in-java/>. 2022.
- [66] Maxiburgos. <https://maxiburgos.medium.com/como-implementar-retrofit-en-nuestra-app-6ba7c>. 2022.
- [67] Stackoverflow. <https://stackoverflow.com/q/53336185>. 2022.
- [68] Node.js. <https://nodejs.org/api/http.html>. 2022.
- [69] Stackoverflow. <https://stackoverflow.com/q/9666030>. 2022.

- [70] Tutorialspoint. https://www.tutorialspoint.com/java_cryptography/java_cryptography_message_digest.htm. 2022.
- [71] Stackoverflow. <https://stackoverflow.com/q/2655972>. 2022.
- [72] Medium. <https://medium.com/bb-tutorials-and-thoughts/how-to-write-simple-nodejs-rest-api-with-core-http-module-dcedd2c1256>. 2022.
- [73] Oracle. <https://docs.oracle.com/javase/tutorial/networking/urls/creatingUrls.html>. 2022.
- [74] Maven. <https://maven.apache.org/>. 2022.
- [75] Gradle. <https://gradle.org/>. 2022.
- [76] Apache Groovy. <https://groovy-lang.org/>. 2022.

Parte B

Anexos

Anexo I

Guía del usuario

Una vez instalada y abierta la aplicación se hace visible el menú de inicio de sesión se puede iniciar sesión o registrarse. Como se muestra en el figura 2.1, en caso de disponer de una cuenta ya creada se introduce el correo electrónico y la contraseña con la que se creó la cuenta para iniciar sesión y en caso de no tener cuenta el botón “Registrarse” ofrece la posibilidad de llenar un formulario y crear así una cuenta.

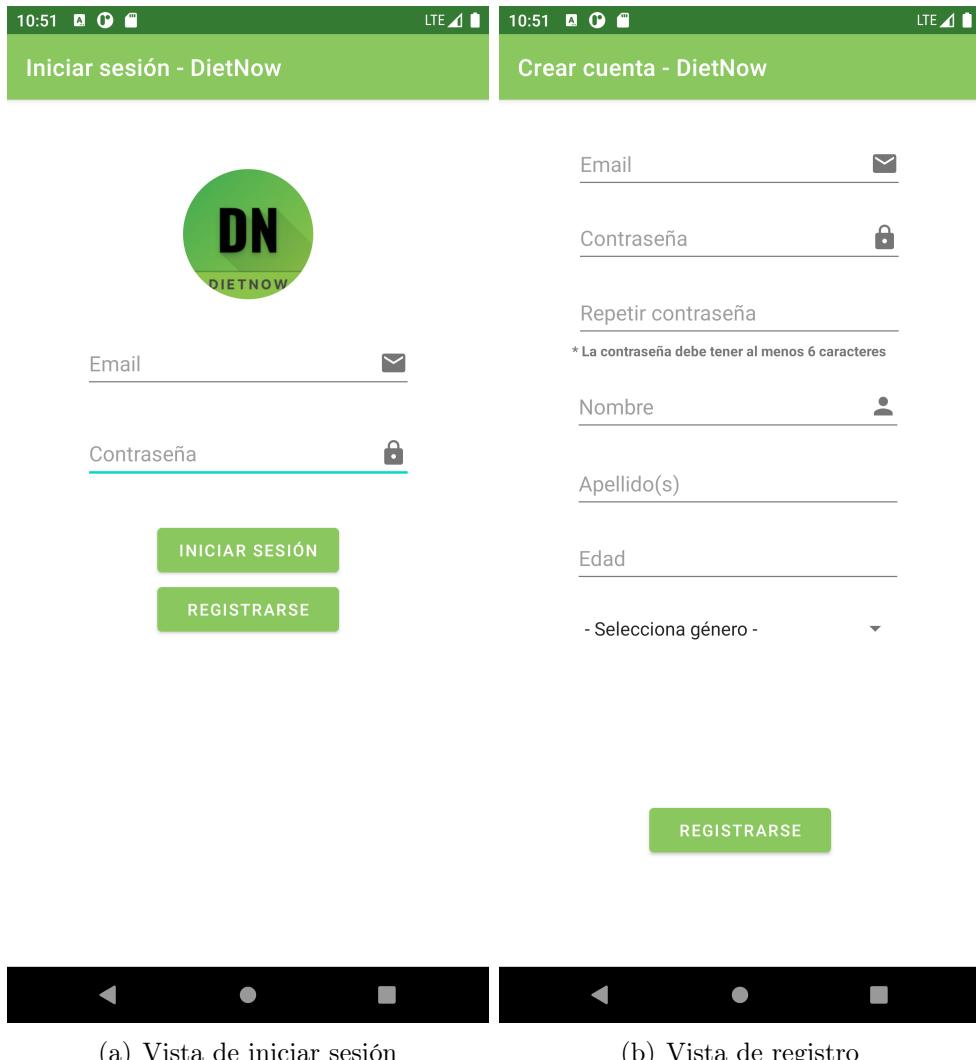


Figura 2.1: Vistas de la aplicación

Cuando se accede a la aplicación lo primero que aparece es el menú principal 2.2, que en función del rol que tenga el usuario verá más o menos opciones, desde este menú se puede acceder a todas las funcionalidades de la aplicación siempre y cuando se tengan los permisos para ello. Los roles son Usuario y Administrador.

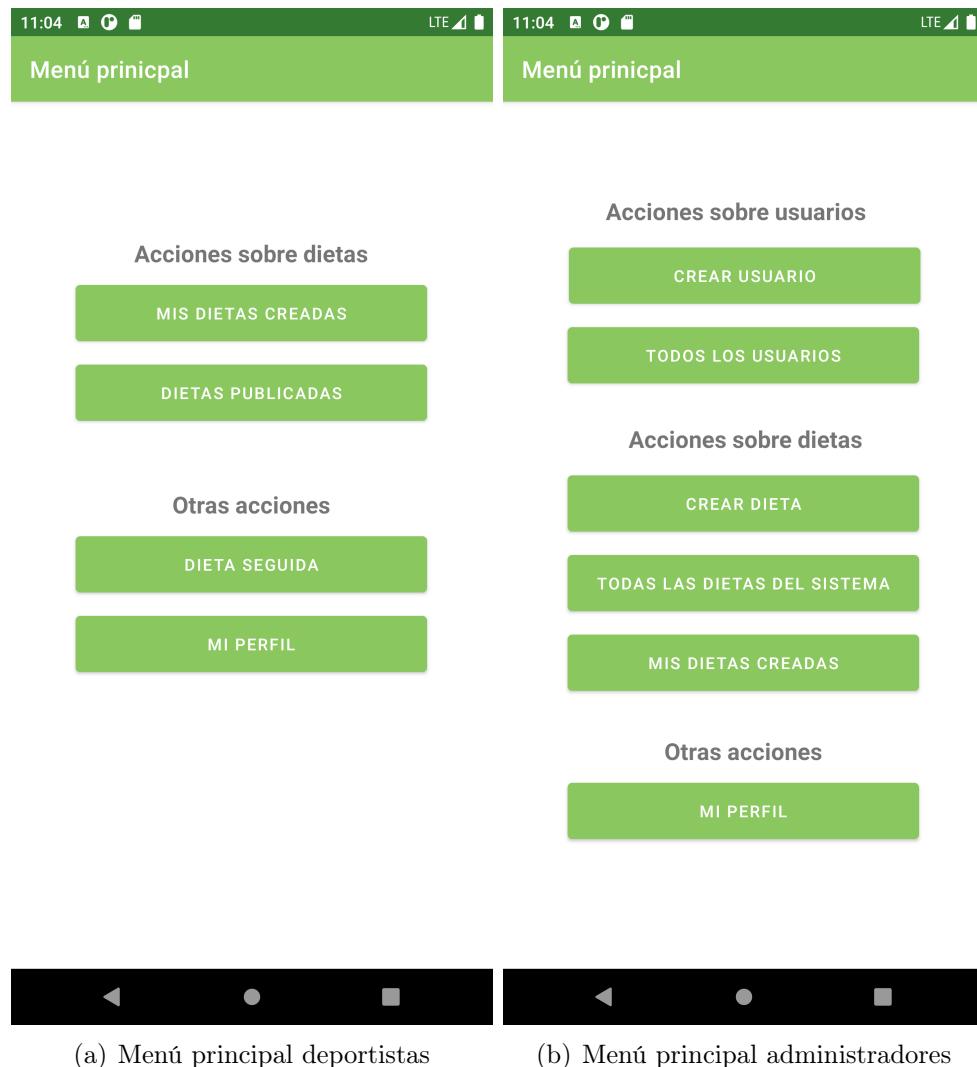


Figura 2.2: Menú principal de la aplicación

2.1. Usuarios

El grueso de la aplicación tiene este rol, los módulos a los que pueden acceder son todos los de dieta y perfil.

2.1.1. Mis dietas creadas

Si el usuario pulsa mis dietas creadas verá un listado de dietas que ha creado, como se muestra en la figura 2.3, si no ha creado ninguna o es un nuevo usuario no verá ninguna dieta. En caso de tener dietas el usuario podrá ver el detalle de cada una pulsando en el botón “Ver dieta”, crear una dieta nueva pulsando el botón con símbolo “+” ubicado en la esquina inferior derecha o filtrar las dietas con un buscador dinámico que se activa al pulsar la lupa que se encuentra arriba del todo.

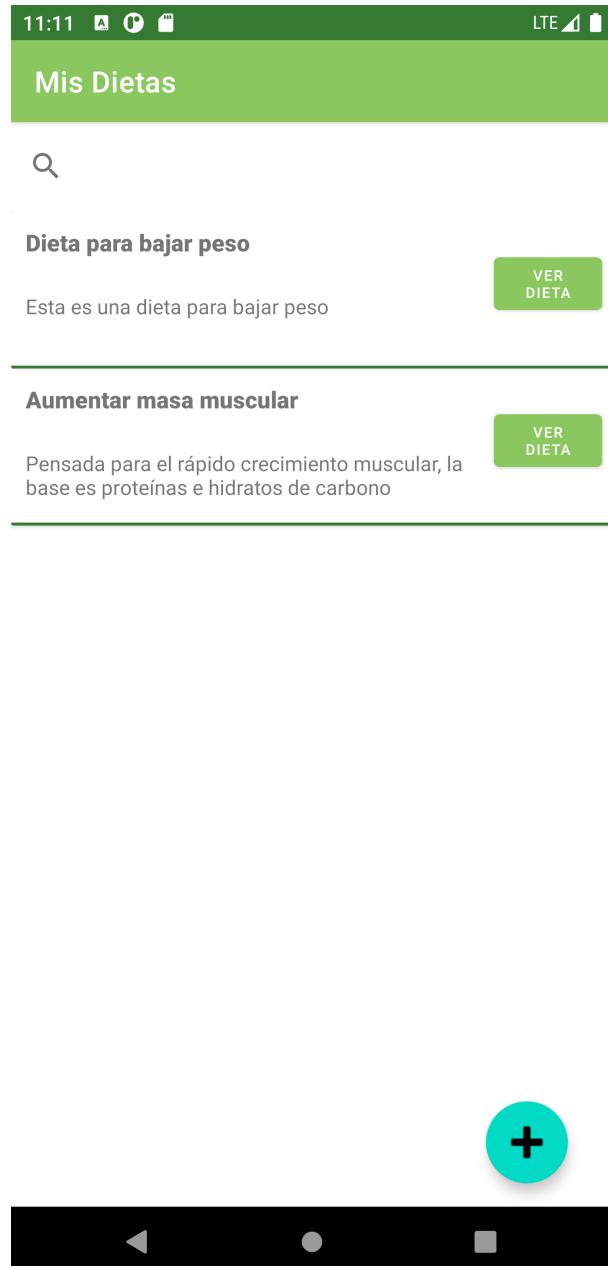


Figura 2.3: Vista detallada de un documento de una dieta

Al presionar “ver dieta” el usuario accede la información detallada de la dieta 2.4 y al ser su autor puede editar o eliminar la dieta, publicarla para el resto de los usuarios o despublicarla si ya estaba publicada. Si la dieta está publicada aparecerá la opción de acceder a los comentarios de la dieta.

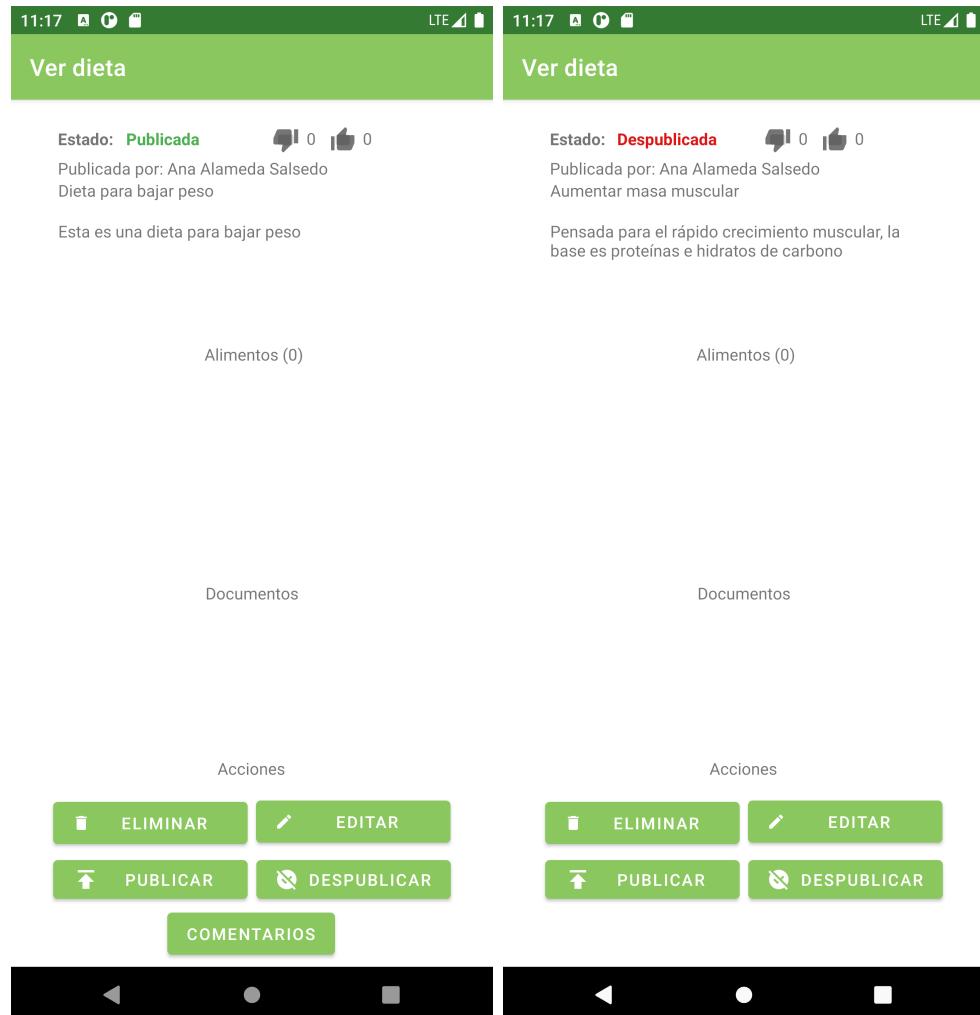
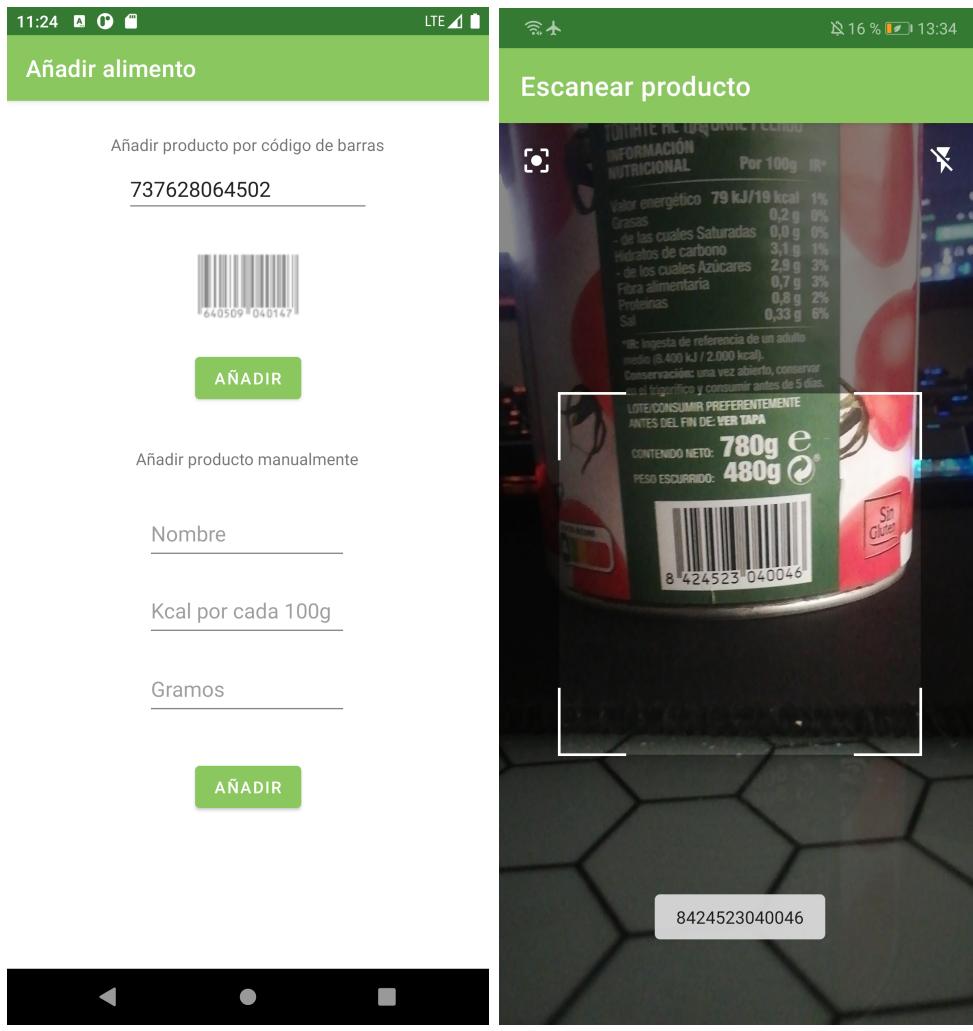


Figura 2.4: Una dieta publicada y despublicada

Si el usuario presiona editar podrá añadir alimentos a la dieta y/o subir documentos en formato PDF con información que el usuario considere necesaria para la dieta como por ejemplo aportar estudios que la respalden.

Para añadir un alimento el usuario dispone de varias formas, como se muestra en la figura 2.5 , desde la cámara, en cuyo caso deberá escanear el código de barras del alimento o manualmente ya sea escribiendo el código de barras o escribiendo los datos del alimento campo por campo.



(a) Añadir manualmente

(b) Añadir con cámara

Figura 2.5: Vista de añadir alimentos a una dieta

2.1.2. Dietas publicadas

Cuando el usuario accede a las dietas publicadas 2.6 podrá visualizar todas las dietas que los usuarios de DietNow han creado y publicado para que estén al alcance de todos, el usuario dispondrá de un buscador dinámico en la parte de arriba para poder filtrar rápidamente las dietas por palabras clave.

Para cada dieta publicada podrá visualizar el número de “Me gusta” que tiene y el número de visualizaciones y al presionar el botón “ver dieta” podrá visualizar el contenido de la dieta además de empezar a seguirla presionando el botón con forma de estrella o abrir la sección de comentarios para leer los comentarios y/o dejar el suyo propio.

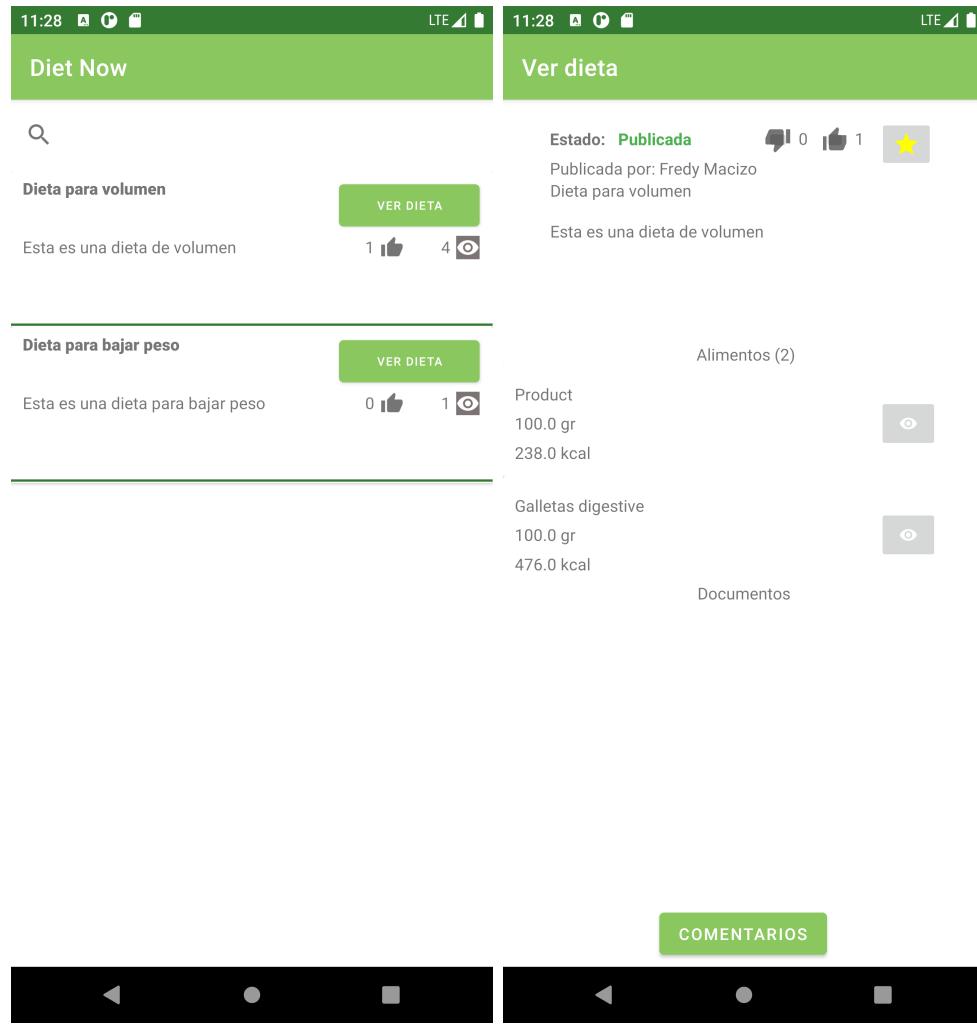


Figura 2.6: Vista dietas publicadas

2.1.3. Dieta seguida

Si el usuario pulsa esta opción verá la dieta que está siguiendo, como se muestra en la figura 2.7, y podrá registrar los alimentos y la cantidad que ha comido en el día actual, podrá ver lo que ingirió a lo largo de la semana y dejar su feedback sobre la dieta ya sea dejando un me gusta, un no me gusta o un comentario, también podrá dejar de seguir la dieta.

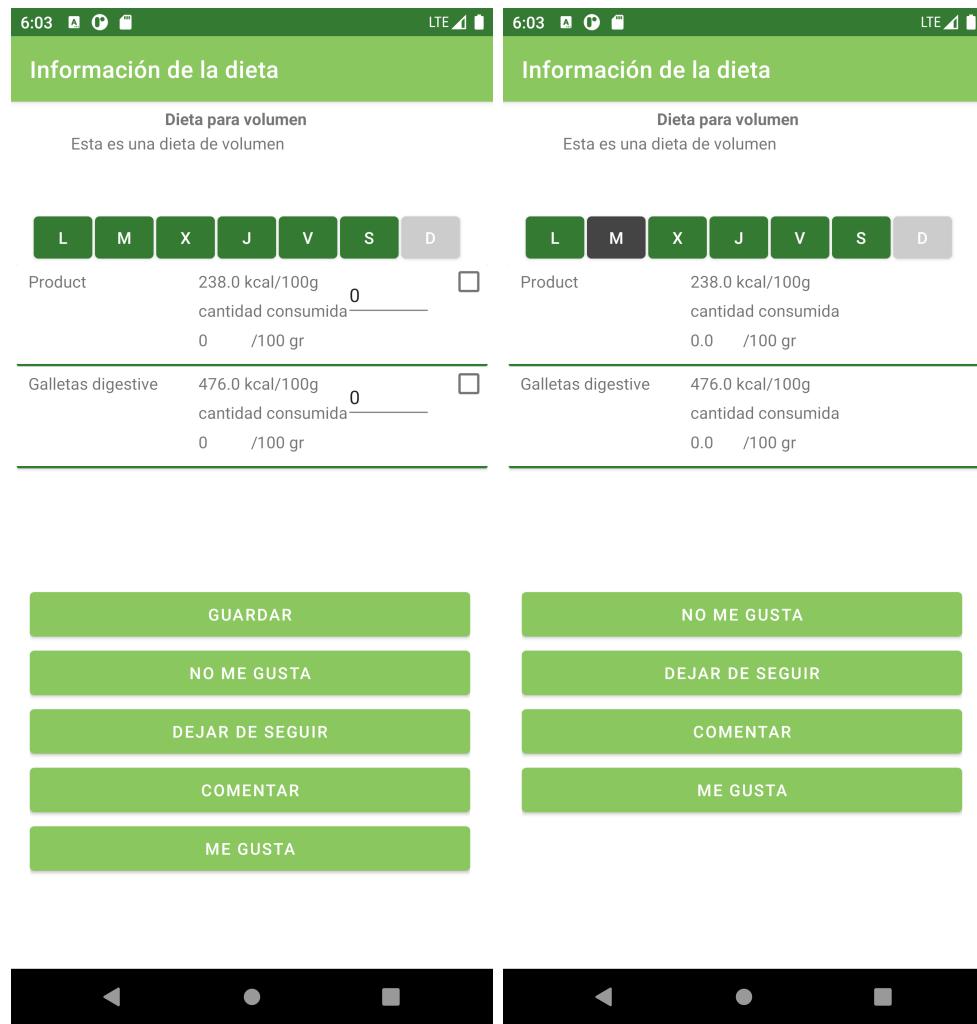


Figura 2.7: Vista de dieta seguida

2.1.4. Ver perfil

Si se selecciona esta opción, el usuario será redirigido a una ventana que mostrará su información personal y las gráficas asociadas a él, como se puede apreciar en la Figura 2.8. Las acciones que puede realizar en esta vista son registrar pasos y/o peso del día actual, ver el historial de dietas seguidas, actualizar sus datos personales, cambiar la imagen de perfil, eliminar el perfil y cerrar sesión.

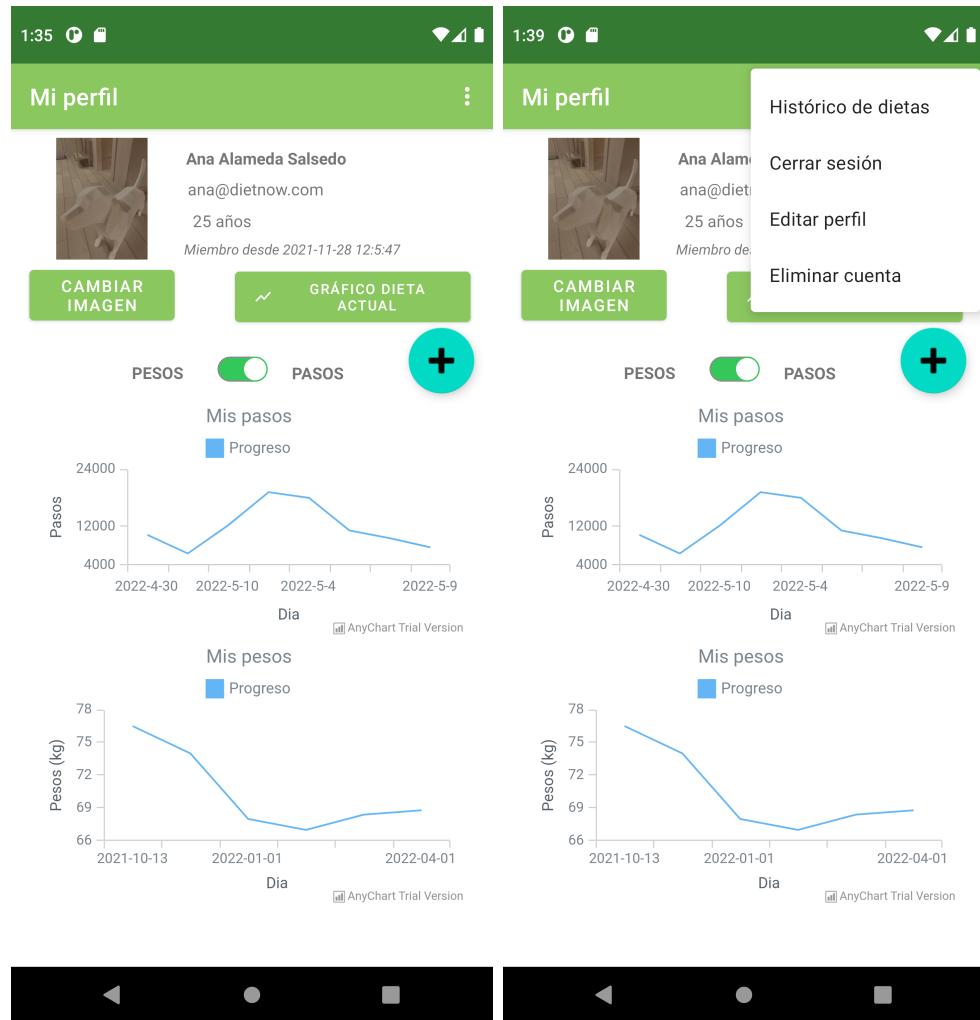


Figura 2.8: Vista del perfil de usuario

2.2. Administradores

Rol pensado para asegurar el correcto funcionamiento de la aplicación, algunas de las funciones de este rol son la gestión de usuarios, creación de dietas predeterminadas y control de contenido.

2.2.1. Crear usuario

Mediante esta opción se accede a un formulario similar al de registro de usuario, como se muestra en la figura 2.9, donde el administrador puede crear una cuenta, la diferencia respecto al formulario de registro es que mientras que en el formulario de registro la cuenta creada siempre tendrá rol usuario, en crear cuenta el administrador puede decidir el rol de la cuenta que está creando.

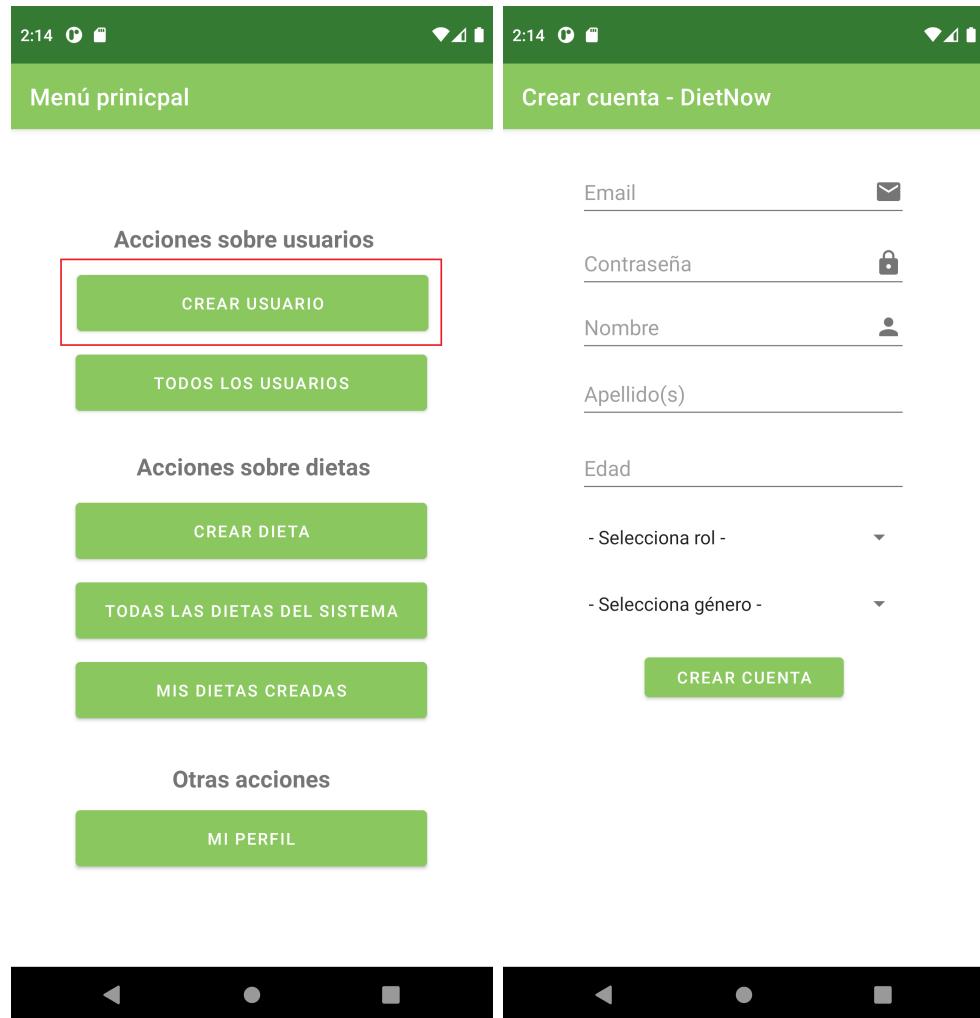


Figura 2.9: Creación de un usuario en la aplicación

2.2.2. Todos los usuarios

Si el administrador selecciona esta opción se abrirá en una nueva ventana un listado con todos los usuarios que están dados de alta en la aplicación, como se muestra en la figura 2.10. En esta pantalla se puede filtrar dinámicamente a los usuarios mediante la lupa de la parte superior y se puede editar o eliminar a los diferentes usuarios de la aplicación.

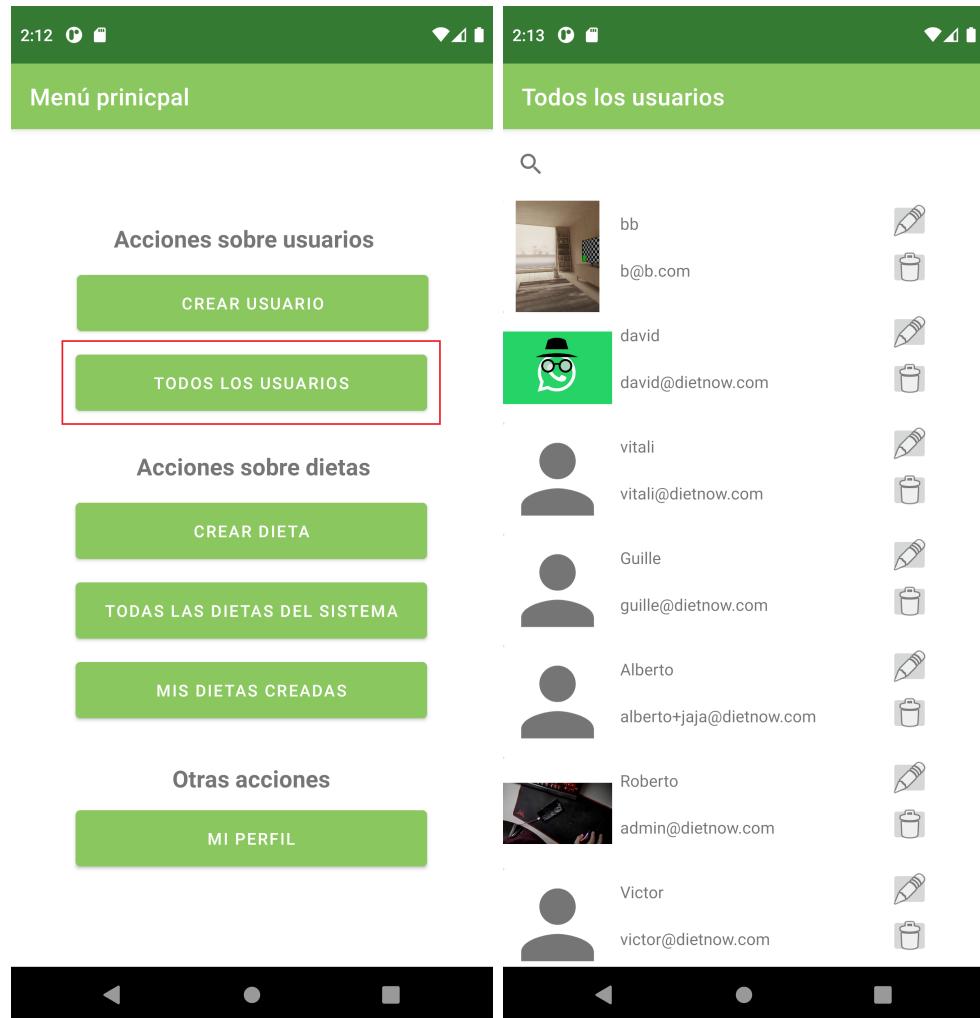


Figura 2.10: Vista de todos los usuarios

2.2.3. Crear dieta

Esta opción abre el formulario de creación de dieta, como se muestra en la figura 2.11, idéntico a cuando en el módulo “Mis dietas creadas” un usuario pulsa el botón circular con el símbolo “+“.

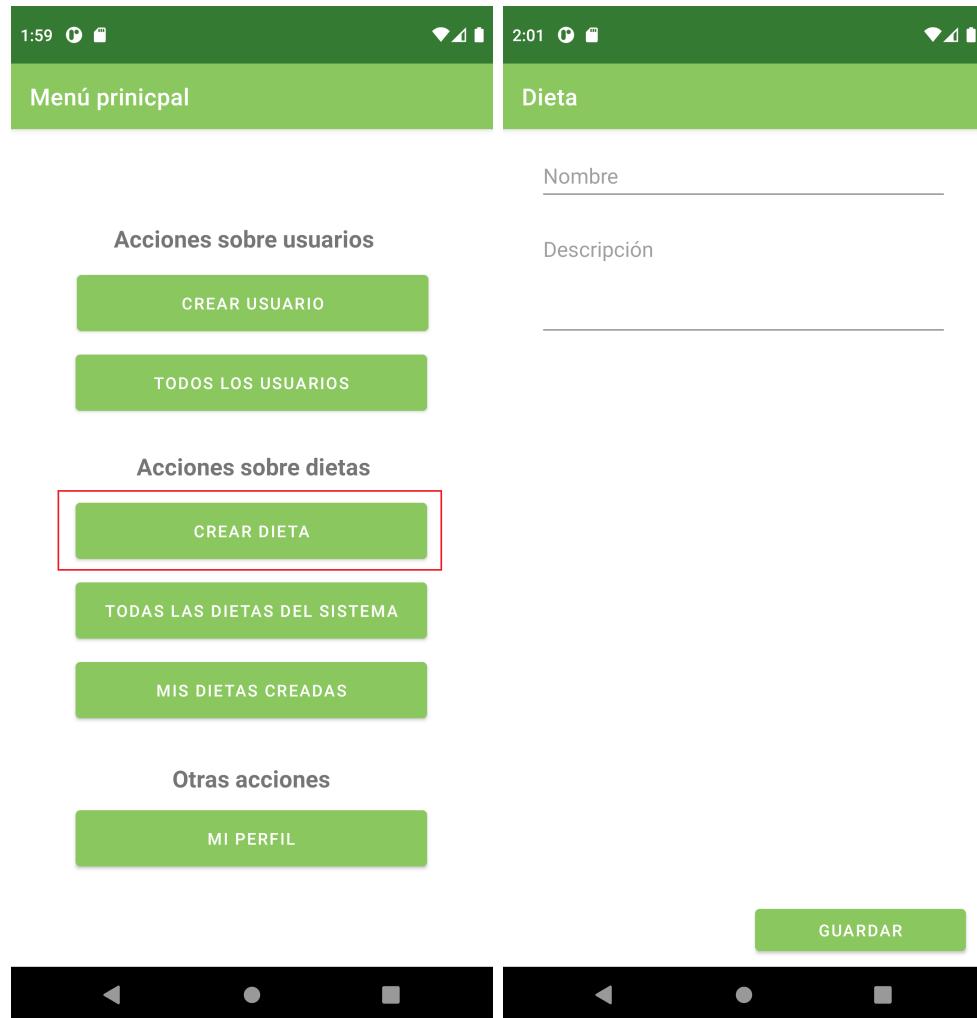


Figura 2.11: Crear una dieta

2.2.4. Todas las dietas del sistema

Mediante esta opción el administrador puede ver todas las dietas de la aplicación 2.12, tanto las publicadas como las privadas, al seleccionar la opción se le mostrará un listado con todas las dietas, el número de me gustas y vistos de cada dieta y su estado, es decir si esta publicada o no, también dispondrá de un filtro dinámico en la parte superior de la vista para filtrar las dietas.

El administrador podrá acceder al detalle de cada dieta pulsando “Ver detalle“ y podrá modificar y/o eliminar la dieta así como cambiar su estado de publicación.



Figura 2.12: Vista de todas las dietas del sistema

2.2.5. Mis dietas creadas

El administrador puede desarrollar esta acción de la misma manera que lo haría un deportista, como se aprecia en el apartado 2.1.1.

2.2.6. Ver perfil

Con esta opción, un administrador accede a su perfil donde se mostrará su información personal y la información del sistema, en esta vista podrá cambiar su imagen de perfil, editar su perfil, cerrar sesión y eliminar cuenta.2.13

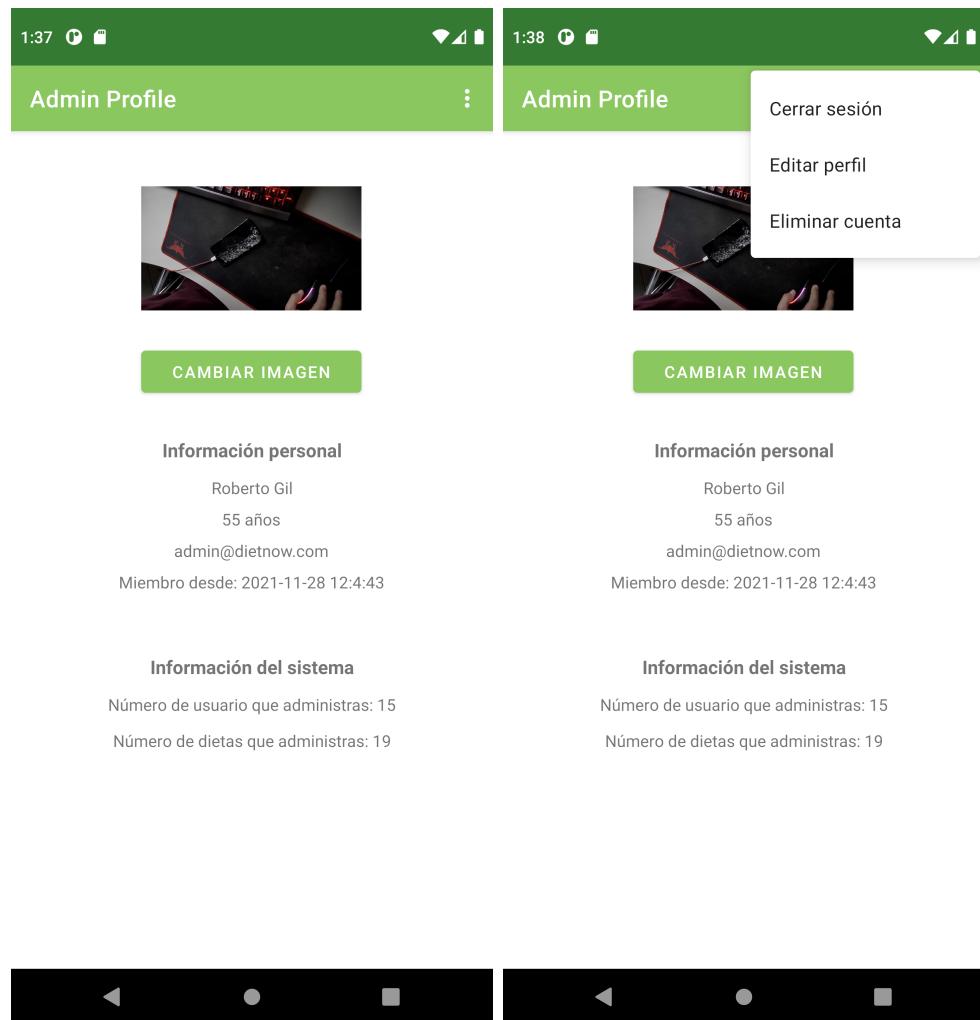


Figura 2.13: Vista de ver perfil de un administrador

Anexo II

Preguntas de la evaluación

En este último anexo se muestran todos los módulos de las preguntas realizadas a los diferentes usuarios que han probado y evaluado la aplicación.

Valoración del usuario (Diet-Now)

Encuesta de valoración del usuario para la aplicación android Diet-Now

[Iniciar sesión en Google](#) para guardar lo que llevas hecho. [Más información](#)

*Obligatorio

Sexo *

Masculino
 Femenino
 Prefiero no decirlo

¿Has tenido alguna dificultad al registrarte? *

Sí
 No

¿El menú principal ofrece las características necesarias? *

Sí
 No
 Tal vez

¿El perfil contiene la información necesaria para desarrollar la actividad? *

Sí
 No
 Tal vez

¿Has creado alguna dieta en la aplicación? *

Sí
 No

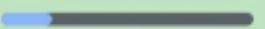
[Siguiiente](#)  Página 1 de 5 [Borrar formulario](#)

Figura 3.1: Preguntas de valoración del usuario

Sección 2 de 5

Creación de dietas

Descripción (opcional)

¿Te parece intuitiva la creación de la dieta? *

Sí

No

Tal vez

¿Mejorarias algo de la creación de la dieta?

Texto de respuesta larga



Figura 3.2: Preguntas de valoración de creación de dietas

Sección 3 de 5

Dieta seguida

Descripción (opcional)

¿Has seguido alguna dieta? *

Sí

No



Figura 3.3: Preguntas de valoración de dieta seguida

Sección 4 de 5

Título de la sección (opcional)

Descripción (opcional)

¿Te parece intuitiva la página de insertar los alimentos correspondientes al día? *

Sí
 No
 Tal vez

¿Te parece útil la sección de comentarios de la dieta? *

Sí
 No
 Tal vez

¿Mejorarias algo de la sección de la dieta seguida?

Texto de respuesta larga

Figura 3.4: Preguntas de comentarios sobre la aplicación

Sección 5 de 5

Valoración final

Descripción (opcional)

Valora del 1 al 10 la aplicación *

1 2 3 4 5 6 7 8 9 10

No me ha gustado nada Me ha encantado

¿Te gustaría que se añadiese alguna función extra a la aplicación?

Texto de respuesta larga

Figura 3.5: Preguntas de valoración final de la aplicación

“Todo lo que tenemos que decidir es qué
hacer con el tiempo que se nos da“
Gandalf

David Fernández Alejo, Vitaliy Savchenko, Carlos Segundo Nieto, Víctor Velasco Arjona

Lunes 30 de mayo de 2022

Ult. actualización 19 de mayo de 2022

\TeX lic. LPPL & powered by **TEFLON** CC-BY-NC-ND

Esta obra está bajo una licencia Creative Commons
“Reconocimiento-NoCommercial-NoDerivs 3.0 España”.

